The following publication Sydora, C., Lei, Z., Siu, M.F.F., Han, S. and Hermann, U. (2021), "Critical lifting simulation of heavy industrial construction in gaming environment", Facilities, Vol. 39 No. 1/2, pp. 113-131 is published by Emerald and is available at https://dx.doi.org/10.1108/F-08-2019-0088

Title: Critical Lifting Simulation of Heavy Industrial Construction in Gaming Environment

Christoph Sydora, Zhen Lei, Ming Fung Francis Siu, SangHyeok Han, Ulrich Hermann

Journal: Facilities – Special Issue: Smart City Facilities and their Management
Purpose – Heavy industrial construction often relies on large mobile cranes to erect equipment and pre-assembled modules. Engineering calculations are required for the lifting analysis where lifting capacity is analyzed to ensure the feasibility of the lifting scenarios. Such engineering calculations are often presented in static formats, e.g. 2D or 3D models. However, it is difficult to help practitioners (e.g. lifting engineers, site crews, and operators) understand the complexity of the lifting process and thus operational decisions are often made intuitively. Therefore, this paper introduces a game-based simulation system to allow for interactive analysis of the lifting process to improve lifting efficiency and safety.
Design/methodology/approach – The proposed method treats the mobile crane as a robot with degree-of-freedoms (DOFs), and the movements are simulated in the Unity game environment. The lifting capacity is calculated dynamically based on the lifting object weight, rigging weight and lifting radius.
Findings – Compared with the 4D visualization, this development has added a dimension of real-time interactive simulation; this allows the users to understand the complexity and feasibility of the lifting process.
Originality/value – The developed prototype has been tested and validated using a real case study from a heavy industrial project with the possibility of generalizing crane lifting configurations.
Keywords: heavy lift plan, game engine, Virtual Reality, mobile crane, heavy industrial construction, construction simulation
Article Type: Research paper

Critical Lifting Simulation of Heavy Industrial Construction in Gaming Environment

Abstract: Heavy industrial construction often relies on large mobile cranes to erect equipment and pre-assembled modules. Engineering calculations are required for the lifting analysis where lifting capacity is analyzed to ensure the feasibility of the lifting scenario. Such engineering calculations are often presented in static formats, e.g. 2D or 3D models. However, it is often difficult for practitioners to understand the complexity of the lifting process and operational decisions are made intuitively. Therefore, this paper introduces a newly developed game-based simulation environment to allow for interactive analysis of the lifting process. The proposed method treats the mobile crane as a robot with degree-of-freedoms (DOFs), and the movements are simulated in the Unity game environment. The lifting capacity is calculated dynamically based on the lifting object weight, rigging weight and lifting radius. Compared with 4D visualization, this development has added a dimension of real-time interactive simulation; this allows the practitioners to understand the complexity and feasibility of the lifting process. The developed prototype has been tested and validated using a real case study from a heavy industrial project with the possibility of generalizing crane lifting configurations.
Keywords: Mobile crane; heavy lift plan; game engine; virtual reality; visualization

1. Introduction and Literature Review

In the province of Alberta, Canada, the oil and gas industry has been the backbone of the local economy. However, productivity has been stagnant due to harsh outdoor working conditions and a lack of a skilled workforce in remote areas. Consequently, modular construction has been predominant in the construction of heavy industrial facilities. For this type of construction, the entire project is broken down into modules that are prefabricated offsite and shipped to the site directly for installation. This helps minimize the exposure of on-site work and improves the heavy equipment utilization rate. PCL Industrial Management Inc. has been a leader in Alberta's heavy industrial construction, and is known forhigh efficiency in construction planning and logistics. Its engineering department has been working closely with academia in the past to develop innovative engineering tools [1–4]. One research initiative is focused to improve heavy lift planning efficiency by automating CAD drafting as well as engineering planning and visualization [5]. This paper discusses the example of a recently developed simulator for heavy lifting using the Unity game engine environment. The simulator can be used to validatecritical lifting plans and also provide safety training. The benefits of this this type of simulation include: (i) allowing users to interact with theheavy lifting environment in three dimensions; (ii) identifying potential lifting hazards and performing collision detection; and (iii) simulating the lifting process to obtain near-optimal lifting scenarios.

Past research in crane operations has seen the development of advanced algorithms to facilitate crane lifts. Automating these types of algorithms is critical to improving efficiency and productivity in planning specific to: (i) crane type and location selection [1,6–8]; (ii) crane path planning and simulation [9–12]; (iii) crane lift visualization [13–16]; and (iv) crane lift engineering design and analysis [15,17–19]. Due to the advancement of digital modeling technologies, crane planning has evolved from a 2D plan to a 3D/4D model with realistic visualization. The practitioners prefer a more direct presentation of the lifting plan rather than previous static designs. In more recent practices, the lifting planning also involves integration with hardware (e.g. laser technologies) to improve safety in the lifting processes [20,21]. The 3D/4D crane lifting simulation is often programmed or pre-calculated, in which case the user cannot interact with the lifting environment and the simulation results are limited to the pre-defined scenarios. Such a challenge is overcome by using game engines to create a dynamic crane simulation, in which the users can control the movement of the crane throughout the lifting process. Figure1 shows this progression of crane lift planning over the past decades due to technological advancements. With the capabilities of dynamic interactions in the gaming environment, construction training environments have then become possible to simulate construction processes for imporved safety and job-related hazard reduction [22–25]. The applications of game technology is also seen as a solution to enhance project management with integration with existing CAD modeling systems (e.g. Building Information Modeling (BIM)) [26]: CAD models are used to develop immersive Virtual Reality (VR) environments [27], and assist with preconstruction planning to achieve better communication and coordination [28]. As VR technologies have matured, the development environment has added new technology such as motion sensors and computer vision to allow for more meaningful interactions between the virtual world and the real world. For example, to understand construction workers' fall risk behavior, an avatar-based system was developed. using the Microsoft Kinect sensor to track real-time motion of construction workers [29]. The development of this type of real-time interactionhas enabled further research to utilize the motion data to improve the performance of the VR models through machine learning algorithms (e.g. reinforced learning methods [29]). Image processing, a computer vision technique, was also used to automatically update the VR gaming environment through machine learning algorithms [30].

Figure 1: Crane lifting planning systems.

Designers from the Architecture, Engineering, and Construction (AEC) and Facility Management (FM) industries utilize tools such as AutoCAD®, Revit®, and 3ds MAX® by Autodesk®to create digital assets [31–34].These assets are then imported to a game development platform to construct the scenes and objects in a virtual environment. A common file type for these type of digital asset is .fbx which sometime requires a file type conversion. Once in the development environment, these objects are controlled by scripts and linked to achieve physical movements of the avatars. Using this process as a basis, this research has developed a game-engine-based simulation for the heavy lifting process that integrates engineering calculations of lifting capacities, and clash detection to detect potential hazards in the lifting process.

2. System Development

2.1 Crane Management System Structure

Thesimulation development originates from a central database where all the project and engineering data is stored. The idea is to provide a central data source for all the engineering planning and provide access to his data from various digital platforms. Applications have been developed specifically for on-site crane management shown in Figure 2 for: crane location selection and optimization [1], lifting sequencing [4], 4D animations from a macro and micro perspective [5], automated client-oriented lift study CAD system. In the central database, three main types of data are stored: (i) the project data: lifting object specification (e.g. dimensions, weight, etc.); (ii) crane data (e.g. crane's dimension, lifting capacity chart, etc.); and (iii) rigging data (e.g. rigging components such as shackles, slings, etc.). The pre-step before this study is the static lift study and 3D crane simulation, which create the crane/module/environment models from CAD libraries. However, legacy models do not interact with each other with physics in a 3D virtual space, which can be achieved through the Unity game engine as discussed in this paper. .

Figure 2: Structure of Crane Management Systems.

2.2 Game Engine Development

The simulation has been developed with the Unity game engine. The Unity software was orginally created for the video game industry to develop computer games running on multiple platforms (i.e iOS, Android, mobile, desktop, console, etc.).. In the Unity environment, the 3D physics engine simulates real-world interactions between virtual objects. The behaviors of objects are controlled by and programmed in .NET script through the application program interface (API). Other game engines, such as Unreal, utilizes API's to control objects. The authors have selected Unity® as the development tool due to its ease of use and large online support community. As shown in Figure 3, in Unity, the foundation for any development is the *Scene*, which contains the environment objects, obstacles and controllable objects that can be added by the user. The *Scene is*therefore, a "global environment" that holds all game items inside. The *Game Objects* are literally the objects in the game environment, which can be a 3D/2D physical object or dummy objects which serve as object containers or scene controllers. In this case, 3D *Game Objects* were primarily used in the development. However, these *Game Objects* are associated with specific properties, and become controllable characters in the game environment (e.g. tree, light, crane boom, etc.). The *Game Objects* can interact with each other, or they can contain multiple *Game Objects* to make themselves nested objects (as described by 1:N relationship in Figure 3). The "Properties" that help the *Game Objects* function in 3D space are called *Components*. For example, the *Transform Component* defines the position, rotation, and scale of a *Game Object* in the 3D *Scene*. Typical *Components* used in our study include: Translation, Rigid Body, Script, Collider, and Joint, which are discussed further in the "Implementation and Case Studies" section.

Figure 3: Unity game engine development structure.

In Unity, *Game Objects* are the base class for all objects in the scene. A game object can represent any real-life 3D entity (e.g. box, tube, etc.). In our development, a crane boom, mast, and crawler are each separate *Game Object*s. They can be grouped together into one parent *Game Object*. However, the crane objects are not able to move, collide or interact in any way without containing components. For a typical crawler crane, the game objects can be defined based on its degree's-of-freedom (DOFs), specifically, the number of independent parameters to define its configuration in a 3D environment. Figure 4 shows a typical crawler crane's DOFs and corresponding game objects used in game development. The defined DOFs also set the movement types of crane in the game environment. For other crane types, the DOFs can be defined differently in implementation.

$$DOFs = \{\alpha, \beta, \gamma, x, z\} \tag{1}$$

Where: α = rotation of the rigging and lifting module; β = boom up and down; γ = rotation of the crane superstructure; x = crawler walking forward and backward; z = hoist up and down.

Figure 4: Degree's-of-Freedom (DOFs) of a typical mobile crane.

Components are the defined attributes of each game object that determines the role of the *Game Object* in the scene. All *Game Objects* inherit a standard object class that contains only the transform component: the object's location, rotation, and scale in a scene. As any *Game Object* can be nested within a parent object, its transform components become relative to its immediate parent game object. Therefore, the global location of an individual game object would be the local position of the object multiplied by the translation matrix of the parent game object. To give a generic example, if an object is the child of an object that's location is rotated by θ about the y-axis and translated by a vector of $[t_x, t_y, t_z]$, then the location of the child object globally can be calculated using the following homogeneous transform:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & t_x \\ 0 & 1 & 0 & t_y \\ -\sin\theta & 0 & \cos\theta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

In Unity, both the local and global locations of the object can be found using the *Transform.localPosition* and *Transform.position* properties of the *Game Object* class respectively. Many different types of components can be added to a single *Game Object* to give it different functions within the *Scene*. For the crane simulation development, some of the key *Component(s)* required are: (i) The *Mesh Component*, which is the "skin" of the object and gives the object its form, although without a *Collider Component* other objects would simply pass through the object; (ii) The *Collider* Component is used to detect the collision between two objects in 3D space. *Collider Components* can take on primitive shapes (such as capsule, sphere, or box colliders) or can use the mesh of an object to take the exact shape of the object (mesh collider). Unity has a built-in Ray Casting mechanism to check collisions and thus reducing the face count of the objects will result in a faster process time; (iii) *Rigidbody Component*, which makes use of the physics simulation, such as gravity. This also impacts the *Collider* functionality. In order for an entire object to react to forces in the *Scene*, each child *Game Object* is given a *Rigidbody Component* which takes in the mass and other kinematic properties of the object; and (iv) A *Joint Component* is a direct connection between two *Game Objects*, specifically the two rigid bodies of each game object. These are added to an object and adjusted such that the combined objects' DOFs accurately match those of the desired structure. By modifying the orientation of the pivot of a joint component, the axis around which the two objects can rotate with respect to the other can be altered.

In addition to the above-mentioned *Components*, custom *Script components* serve as means of controlling *Game Objects* over short iterations in a way that cannot be done by other predefined components alone (e.g. movement, rotation, etc.). All scripts inherit a base class *MonoBehaviour*. *MonoBehaviour* contains predefined methods that are executed automatically depending on the function of the method. There are four main methods of *MonoBehaviour* and these are listed in Table 1. When certain events are triggered, an object's behavior and properties can be programmatically altered as needed in the scene. For example, when two objects colliders come into contact, an event would be triggered and forces or components can be applied to the objects in order to mimic any type of effect, such as an explosion or a fusion of the objects using joints. Using events that can be set to trigger over certain time intervals allows adding controls to the game object by checking keys that are pressed. For example, when a keyboard key is pressed the object will respond to the corresponding action such as booming up or down. Scripting is also essential for network managing and controlling the network flow and synchronization in the scene.

Table 1: Sub-classes of Unity Script.

3. Implementation and Case Studies

3.1 Model Import and Script

The crane lift study is broken down into separate objects suitable to import into Unity as *.fbx* files. The objects include crane mats, crane boom, hoist cable linking rigging to boom, crane's super-lift attachment, crane crawler, mast and derrick, surrounding environment and obstacles, lifting module, and delivery truck. After importing, required *Components* are added, and the *Game Objects* are arranged and properly aligned with respect to each other to re-create the real-world lift scenario (a hierarchy is created to link and group the related *Game Objects* together). Figure 5 shows the Unity interface with the imported model. The interface contains the edited window and the game play testing window (where camera points at). The *hierarchy* defines the *Game Objects* structure, and in this study, a main Game Object "Crane" is created to include all crane objects including the crane crawler track, the derrick mast, super-lift counterweight, rigging, and cables. Instead of using separate scripts to control individual crane objects, a master script can be used to control all sub-objects through the main "Crane" object. The hierarchy is also defined by the DOFs of the crawler crane. The crane crawler track is at a higher level in the hierarchysince the movement of the crawler track dictates the movements of the other crane superstructures (e.g. boom, mast, etc.). For example, *boom-up* does not trigger the movement of the crawler track. All crane objects link with each other through the Unity component *Joint*.

Figure 5: Unity Interface with Imported 3D Model.

Custom *Scripting* was used to control crane components, specifically the user controls and subsequent moments and as triggers for lift simulation feedback. In this development, *Scripts* can be categorized into the following kinds: (i) crane movement script which control the crane movements; (ii) camera control script that allows the user to move the camera and view the project from difference perspectives; and (iii) collision/complete event trigger script which detects any collisions in the lifting process and completes the lifting once the object reaches its set location. For the crane movement, there are two types of movements: translation and rotation, which can be controlled by *GameObject.Transform* function, a built-in parameter of Unity *Game Object*. The *GameObject.Transform* allows the user to access the object's 3D position, rotation angle, and scale. For example, for crane crawler forward-movement can be controlled through the crawler's *rigidbody* (`private Rigidbody crawlerRB`), and when the `KeyCode` CrawlerForward is pressed, the updated crawler position can be redefined by a new `Vector3` in FixedUpdate():

```
crawlerRB.transform.position = new Vector3
(crawlerRB.transform.position.x + crawlerObj.transform.forward.x,
crawlerRB.transform.position.y + crawlerObj.transform.forward.y,
crawlerRB.transform.position.z + crawlerObj.transform.forward.z);
```

During the lifting process, constant checking forcollisions between the crane and the surrounding environment is done through the *Collider* object. *Collider* objects can be defined and checked with each other in the `void OnCollisionStay(Collision collision)` function through game objects' name tags. When a collision occurs, this method is triggered and, depending on the objects involved, scene objects can be modified e.g. displaying a warning symbol to the user or simulating a crash event.

Crane components that required a little more elaborate control were the cables from the super-lift to the boom and from the the boom to the lifting module and rigging. This is because when the boom moves, the total cable length must be maintained. In our implementation, we opted for a simplistic two-piece cable, each simulated as a long thin cylinder. Therefore, the actual boom and super-lift control are dependent on the cable rescaling rather than their movements in space. There are more elaborate methods that could have been employed to improve the realism of the swing such as using a single series of minuscule segments or a kinematic chain, however, using a single cable with full swing range was sufficient for the level of detail in this research. Importing additional objects from the Unity Asset Store or other third-party object vendors is possible for this scenario; however, the scale must then be maintained such that all objects follow the Unity scene unit which corresponds to 1 meter.

3.2 Crane Lifting Simulation in Unity Environment

Crane lifts can be categorized into two types: (i) the non-critical lift; and (ii) the critical lift. The critical lift requires engineering design before lifting due to site congestion and/or high lifting capacity (e.g. more than 80% of the crane's lifting capacity). In the presented case, a real critical lift is used for testing the developed game engine simulation. As shown in Figure 6, a LR-1600 crane with buggy super-lift attachment is used to perform this lift. The total weight is 114,017 lbs (51.7 Te) (including module load weight, load block weight, rigging weight, and auxiliary ball and runner). At maximum radius of 115 ft (35.1m) and 88.4% of crane capacity is used given the manufacturer's chart capacity is stated to be 128,970 lbs (58.5 Te). Due to its high capacity and congested surrounding, this lift is considered a critical lift. In general, the crane's lifting capacity is calculated by the following equation:

$$C_{lifting} = \frac{W_{Module} + W_{Rigging} + W_{LoadBlock} + W_{Additional}}{W_{Chart}} \times 100\% \quad (3)$$

Where: $C_{lifting}$ = crane lifting capacity; W_{Module} = module weight; $W_{Rigging}$ = Rigging weight; $W_{LoadBlock}$ = Load block weight; $W_{Additional}$ = Additional weight (e.g. auxiliary ball and runner); W_{Chart} = Chart allowable lifting capacity.

As shown in Figure 6, the module is picked up from a delivery truck and swung clockwise to the set position. The static lift study in Figures 5 and 6 do not give a clear understanding of potential collision that the lifting object may encounter with the grey existing structures at the final set position. The crane operator and other on-site staff would thus have to "imagine" the trajectory for the lifting module to be maneuvered without any collision. An elevation view is provided to show the potential collision at the final position between the lifting module and the surrounding structures (Figure 7). The elevation view also shows merely a static scene where the lifting module is at its final set location, which provides the clearance checking. In Unity, the lifting scenario is created as shown in Figure 8. Through the keyboard control, the user is able to maneuver the lifting object to its set location, during which the lifting capacity is checked at each time frame as well as having continuous collision detection (as in Figure 9 where the lifting radius is 130.65ft with a lifting capacity of 30.77%). Once the lifting object reaches its set location (Figure 10), the Script that checks lift progress is triggered and a "Complete" sign is displayed. An example list of allowed movements along with the control keys that the user can perform is:

W - Crane Forward
S - Crane Backward
A - Crane Left
D - Crane Right
Q - Main Camera Left
E - Main Camera Right
Y - Boom Left
U - Boom Right
I - Boom Forward
O - Boom Backward
G - Rigging Halt
H - Rigging Rotate CounterClock
J - Rigging Rotate Clockwise
K - Rigging Up
L - Rigging Down

Figure 6: Plan view of a critical heavy lift study.

Figure 7: Elevation view of a critical heavy lift study.

Figure 8: Lifting Scenario in Unity Environment.

Figure 9: Crane lifting module simulation.

Figure 10: Crane lifting complete at the module set location.

To summarize, the steps required to recreate the lift simulation are as follows:
(i) import all crane, module, and environment objects;
(ii) arrange all objects into their starting positions and hierarchies;(iii) add each object's physics components, namely the rigid body, collider, and joints between the connected objects;
(iv) for dynamic scene objects, add additional controller scripts for movement and lift simulation progress (based on distance from current to desired goal state);
(v) place Camera objects in the crane's operator location and additional locations in the scene as necessary. The following section will elaborate on this for multiple users.

3.2.1 Multi-player Simulation

With the addition of Unity's NetworkManager class, we were able to achieve a multiple player environment. In this environment, three players (Fig. 11) can simultaneously interact and collaborate to complete the lift. This simulates a more realistic lifting scenario compared to a single player system. Limitations of the views of the crane operators are shown in Figure 12 where it is difficult to observe the lifted object's location, while also considering that there are many blind spots in the process of lifting. With the aid of one or more signalers (Figure 13 shows where they are located on the ground and on the platform of the obstructions), instant feedback of the lifting is given to the operator, who can then can make reasonable and safer decisions. Figures 12 and 13 are taken from the same moment where the crane sets the lifted object at its destination. From Figure 12 it is unlikely to capture the complete picture of the lifting environment and the clearance from a collision is questionable. In Figure 13, the signal persons are able to ensure the safety of the lift and provide valuable feedback to the operator.
In the Unity environment, a High-Level API (HLAPI) server authoritative system is used for building multiplayer games. It allows one of the players to be the client and server at the same time, which in our case will be the crane operator [33].As such, no dedicated server is needed and other players can simply connect to the server (the crane operator's machine) through its Internet Protocol address (IP address). The *Network Identity* and *Network Transform* Components are used and attached to the crane objects to ensure object synchronization between each user's scenes. The scripts for controlling these two components are:

```
private NetworkManager networkManager;
private InputField ipInput;
private InputField portInput;
/// <summary>
    /// When game starts up
    /// </summary>
    void Start()
    {
        Debug.Log("NetManagerStart");
        networkManager = GameObject.Find("NetworkManager").GetComponent<NetworkManager>();
        ipInput = GameObject.Find("IPInputField").GetComponent<InputField>();
        portInput = GameObject.Find("PortInputField").GetComponent<InputField>();
    }

    /// <summary>
    /// when server starts up:
    /// </summary>
```

```
public override void OnStartServer()
{
    Debug.Log("NetManagerOnStartServer");
    networkManager.serverBindAddress = ipInput.text;
    networkManager.networkAddress = ipInput.text;
    networkManager.networkPort = Convert.ToInt32(portInput.text);
}
```

Figure 11: Unity Engine Multiple Player Scenario.

Figure 12: Multiple Player – Crane Operator Perspective.

Figure: 13 Multiple Player – Signal Person Perspective.

4. Generalization of Crane and Lift Configuration

In order for the lift simulation to be irrespective of a crane model, lift module, or lift environment, a certain amount of data beyond the geometrical is required for each crane component. Loading a full crane 3D model into a Unity scene would be problematic, as each of its components (boom, crawler, super-lift, etc.) would not be able to move independently of each other. Thus, as with our implementation, each crane component must be loaded separately to the scene. For our implementation, the reconstruction of the full crane from its components was performed manually, however, if connection information for each component was provided, the full crane object could be automatically reconstructed. The requirements for such as reconstruction are as follows: (i) Each component, in addition to its own metadata such as type and mesh geometry, must have a joint location, degree of freedom (rotation axis and/or movement plane) and movement speed, the component type that must be connected on the other end, and the default or starting configuration angle. This would allow the automated reconstruction of the crane to piece together each component starting with the base crawler and building up one by one. (ii) Next, the crane would need the connecting cables to be attached. This is a separate step done after the crane is constructed, as the cables need to be shortened or lengthened based on the default setup. In addition, the cable objects extend and contract relative to each other. For example, if the boom lowers, the cable connecting the superlift to the boom extends, resulting in a contraction of the cable connecting the rigging to the boom, since the total length of the cables combined must be maintained. Therefore, to automatically model a cable, there must be a predefined start and end point to the cable with intermediate points sequentially ordered. For the case study, this corresponded to the start point being the superlift followed by the tip of the boom and finally ending at the rigging attachment. Therefore, a single controller can control all cable manipulations and maintain consist cable lengths. Once each of the components have been connected and the cables have been placed, each of the possible movements, based on the degrees of freedom, can then be assigned a controller key or button.By utilizing BIM within the lifting environment and the properties of each of the objects and the selected rigging, the possible lifting points of the lifting module can be determined such that the points on the lift module and rigging align. (iii) Finally, based on the mesh of each of the objects in the scene, colliders can be attached with a script alerting the user of collisions between environment objects and crane components. Each of the vantage points can be based on freely moving cameras, with one being fixed to the crane's operation cabin.

5. Conclusions and Future Work

In this paper, a crane simulator has been developed using the Unity game engine. In the developed system, the crane is treated as a robot with DOFs and considers realistic lifting capacities. The a virtual physics engine was implemented to help detect collisions and determine the completion of a lift. Imported from CAD as *.fbx* files, the rigid crane components are linked and controlled by the script with colliders added for clash detection. In the methodology and implementation, algorithms and related API's were introduced so that other researchers/practitioners can refer to them in future work or implementation. Two real case studies were provided to validate the proposed method and efficiency of the simulation. In the first case, the single-player crane operation is presented with the lifting capacity calculation and lifting radius during the lift. From the single-player mode, it can be concluded that blind spots can impede the crane operator ability to direct the lift without assistance from the signal persons. In case two, a multi-player crane scenario is developed and presented using the *NetworkManager* component in Unity. The HLAPI system in Unity allows the crane operator-player to host the machine and avoid a dedicated server for multiplayer purposes. This development provides a systematic and effective approach to provide an interactive simulation for crane lifting scenarios. It helps the users to identify safety hazards and virtually rehearse the lifting process. The testing can also allow the workers to become more efficient. For example, through iterations of practice and discussion, reasonable lifting paths can be arranged to avoid unnecessary movements and increase the ease of lifting. The VR development can be introduced to construction on-site, pre-lift meetings as an instructive tool for crews to enhance their understanding of the construction execution plans. Additionally, the signal persons can virtually practice on-site coordination of the lifting process. Although it is a prototype, the authors think there is a positive impact on crane lifting safety and training and will extend the work in that direction. The authors have also realized there is a limitation that each lifting scenario requires manual adjustment of the model and setup. This can be mitigated to some extent in the future through automation using API codes and reduce the human effortto create the simulation. Another possible solution could be using Cloud-based BIM metadata to synchronize CAD models and the VR environment [35]. Meanwhile, commercial VR equipment such as Oculus and Vive can be used for further development of an immersive virtual environment.

6. Acknowledgements

The authors would like to acknowledge all the participants in the research, particularly our industry collaborator PCL Industrial Management Inc. We would also like to express our appreciation for the support received from the Hole School of Construction Engineering, and the Nasseri School of Building Science and Engineering at the University of Alberta. Support from the Off-site Construction Research Centre (OCRC) at the University of New Brunswick is also acknowledged.

References:

[1] U. Hermann, A. Hendi, J. Olearczyk, M. Al-Hussein, An Integrated System to Select, Position, and Simulate Mobile Cranes for Complex Industrial Projects, in: Constr. Res. Congr. 2010, American Society of Civil Engineers, Reston, VA, 2010: pp. 267–276. doi:10.1061/41109(373)27.

[2] Z. Lei, H. Taghaddos, U. Hermann, M. Al-Hussein, A methodology for mobile crane lift

path checking in heavy industrial projects, Autom. Constr. 31 (2013) 41–53. doi:10.1016/j.autcon.2012.11.042.
[3] Z. Lei, H. Taghaddos, J. Olearczyk, M. Al-Hussein, U. Hermann, Automated Method for Checking Crane Paths for Heavy Lifts in Industrial Projects, J. Constr. Eng. Manag. (2013) 04013011-1–9. doi:10.1061/(ASCE)CO.1943-7862.0000740.
[4] H. Taghaddos, Asce, U. Hermann, S. Abourizk, M. Asce, Y. Mohamed, Simulation-Based Multiagent Approach for Scheduling Modular Construction, J. Comput. Civ. Eng. 28 (2014) 263–274. doi:10.1061/(ASCE)CP.1943-5487.0000262.
[5] Z. Lei, H. Taghaddos, S. Han, A. Bouferguène, M. Al-hussein, U. Hermann, From AutoCAD to 3ds Max : An automated approach for animating heavy lifting studies, Can. J. Civ. Eng. 1 (2016) 5404. doi:10.1139/cjce-2014-0313.
[6] C. Huang, C.K. Wong, C.M. Tam, Optimization of tower crane and material supply locations in a high-rise building site by mixed-integer linear programming, Autom. Constr. 20 (2011) 571–580. doi:10.1016/j.autcon.2010.11.023.
[7] L.-C. Lien, M.-Y. Cheng, Particle bee algorithm for tower crane layout with material quantity supply and demand optimization, Autom. Constr. 45 (2014) 25–32. doi:10.1016/j.autcon.2014.05.002.
[8] H. Safouhi, M. Mouattamid, U. Hermann, a. Hendi, An algorithm for the calculation of feasible mobile crane position areas, Autom. Constr. 20 (2011) 360–367. doi:10.1016/j.autcon.2010.11.006.
[9] H.R. Reddy, K. Varghese, Automated Path Planning for Mobile Crane Lifts, Comput. Civ. Infrastruct. Eng. 17 (2002) 439–448. doi:10.1111/0885-9507.00005.
[10] Y. Lin, D. Wu, X. Wang, X. Wang, S. Gao, Lift path planning for a nonholonomic crawler crane, Autom. Constr. 44 (2014) 12–24. doi:10.1016/j.autcon.2014.03.007.
[11] Y.-C. Chang, W.-H. Hung, S.-C. Kang, A fast path planning method for single and dual crane erections, Autom. Constr. 22 (2012) 468–480. doi:10.1016/j.autcon.2011.11.006.
[12] M. Ali, N. Babu, K. Varghese, Collision free path planning of cooperative crane manipulators using genetic algorithm, J. Comput. Civ. Eng. 19 (2005) 182–193. http://ascelibrary.org/doi/abs/10.1061/(ASCE)0887-3801(2005)19:2(182) (accessed May 28, 2014).
[13] J.R. Juang, W.H. Hung, S.C. Kang, SimCrane 3D+: A crane simulator with kinesthetic and stereoscopic vision, Adv. Eng. Informatics. 27 (2013) 506–518. http://linkinghub.elsevier.com/retrieve/pii/S1474034613000529 (accessed May 23, 2014).
[14] C. Zhang, A. Hammad, Improving lifting motion planning and re-planning of cranes with consideration for safety and efficiency, Adv. Eng. Informatics. 26 (2012) 396–410. doi:10.1016/j.aei.2012.01.003.
[15] Y. Lin, D. Wu, X. Wang, X. Wang, S. Gao, Statics-based simulation approach for two-crane lift, J. Constr. Eng. Manag. 138 (2012) 1139–1149. http://ascelibrary.org/doi/abs/10.1061/(ASCE)CO.1943-7862.0000526 (accessed May 28, 2014).
[16] W.C. Hornaday, C.T. Haas, J.T. O'Connor, J. Wen, Computer-aided planning for heavy lifts, J. Constr. Eng. Manag. 119 (1993) 498–515.
[17] S. Hasan, M. Al-hussein, D. Ph, U.H. Hermann, H. Safouhi, Interactive and Dynamic Integrated Module for Mobile Cranes Supporting System Design, (2010) 179–186.
[18] S. Hasan, A. Bouferguene, M. Al-Hussein, P. Gillis, A. Telyas, Productivity and CO2 emission analysis for tower crane utilization on high-rise building projects, Autom. Constr.

31 (2013) 255–264. doi:10.1016/j.autcon.2012.11.044.

[19] D. Wu, Y. Lin, X. Wang, S. Gao, Algorithm of crane selection for heavy lifts, J. Comput. Civ. Eng. 25 (2011) 57–65. http://ascelibrary.org/doi/abs/10.1061/(ASCE)CP.1943-5487.0000065 (accessed May 28, 2014).

[20] G. Lee, H.-H. Kim, C.-J. Lee, S.-I. Ham, S.-H. Yun, H. Cho, B.K. Kim, G.T. Kim, K. Kim, A laser-technology-based lifting-path tracking system for a robotic tower crane, Autom. Constr. 18 (2009) 865–874. doi:10.1016/j.autcon.2009.03.011.

[21] S. Hwang, Ultra-wide band technology experiments for real-time prevention of tower crane collisions, Autom. Constr. 22 (2012) 545–553. http://linkinghub.elsevier.com/retrieve/pii/S0926580511002226 (accessed May 28, 2014).

[22] I.M. Rezazadeh, X. Wang, M. Firoozabadi, M.R. Hashemi Golpayegani, Using affective human-machine interface to increase the operation performance in virtual construction crane training system: A novel approach, Autom. Constr. 20 (2011) 289–298. doi:10.1016/j.autcon.2010.10.005.

[23] J. Goulding, W. Nadim, P. Petridis, M. Alshawi, Construction industry offsite production: A virtual reality interactive training environment prototype, Adv. Eng. Informatics. 26 (2012) 103–116. doi:10.1016/j.aei.2011.09.004.

[24] S.P. Smith, D. Trenholme, Rapid prototyping a virtual fire drill environment using computer game technology, Fire Saf. J. 44 (2009) 559–569. doi:10.1016/j.firesaf.2008.11.004.

[25] H. Guo, H. Li, G. Chan, M. Skitmore, Using game technologies to improve the safety of construction plant operations, Accid. Anal. Prev. 48 (2012) 204–213. doi:10.1016/j.aap.2011.06.002.

[26] X. Wang, P.E.D. Love, M. Jeong, C. Park, C. Sing, L. Hou, A conceptual framework for integrating building information modeling with augmented reality, Autom. Constr. 34 (2013) 37–44. doi:10.1016/j.autcon.2012.10.012.

[27] J. Whyte, N. Bouchlaghem, A. Thorpe, R. McCaffer, From CAD to virtual reality: Modelling approaches, data exchange and interactive 3D building design tools, Autom. Constr. 10 (2000) 43–45. doi:10.1016/S0926-5805(99)00012-6.

[28] A.F. Waly, W.Y. Thabet, A Virtual Construction Environment for preconstruction planning, Autom. Constr. 12 (2003) 139–154. doi:10.1016/S0926-5805(02)00047-X.

[29] Y. Shi, J. Du, C.R. Ahn, E. Ragan, Impact assessment of reinforced learning methods on construction workers' fall risk behavior using virtual reality, Autom. Constr. 104 (2019) 197–214. doi:10.1016/j.autcon.2019.04.015.

[30] F. Pour Rahimian, S. Seyedzadeh, S. Oliver, S. Rodriguez, N. Dawood, On-demand monitoring of construction projects through a game-like hybrid application of BIM and machine learning, Autom. Constr. 110 (2020) 103012. doi:10.1016/j.autcon.2019.103012.

[31] N.O. Nawari, BIM Standard in Off-Site Construction, J. Archit. Eng. 18 (2012) 107–113. doi:10.1061/(ASCE)AE.1943-5568.0000056.

[32] Z. Hu, J. Zhang, BIM- and 4D-based integrated solution of analysis and management for conflicts and structural safety problems during construction: 2. Development and site trials, Autom. Constr. 20 (2011) 155–166. doi:10.1016/j.autcon.2010.09.013.

[33] B. Akinci, M. Fischer, J. Kunz, Automated Generation of Work Spaces Required by Construction Activities, J. Constr. Eng. Manag. 128 (2002) 306–315. doi:10.1061/(ASCE)0733-9364(2002)128:4(306).

[34] S. Zhang, J. Teizer, J.K. Lee, C.M. Eastman, M. Venugopal, Building Information

Modeling (BIM) and Safety: Automatic Safety Checking of Construction Models and Schedules, Autom. Constr. 29 (2013) 183–195. doi:10.1016/j.autcon.2012.05.006.
[35] J. Du, Z. Zou, Y. Shi, D. Zhao, Zero latency: Real-time synchronization of BIM data in virtual reality for collaborative decision-making, Autom. Constr. 85 (2018) 51–64. doi:10.1016/j.autcon.2017.10.009.
[33] https://docs.unity3d.com/Manual/UNetUsingHLAPI.html (accessed on Sep 4th 2017)