An Initial Julia Simulation Approach to Material Handling Operations from Motion Captured Data

M. Hernandez
Ingram School of Engineering
Texas State University
San Marcos, USA
mah338@txstate.edu

D. Valles
Ingram School of Engineering
Texas State University
San Marcos, USA
dvalles@txstate.edu

D. C. Wierschem
McCoy College of Business
Texas State University
San Marcos, USA
dw50@txstate.edu

R. M. Koldenhoven
Department of Heath & Human Performance
Texas State University
San Marcos, USA
rmr214@txstate.edu

G. Koutitas
Ingram School of Engineering
Texas State University
San Marcos, USA
george.koutitas@txstate.edu

F. A. Mendez
McCoy College of Business
Texas State Unibersity
San Marcos, USA
fm16@txstate.edu

S. Aslan
Ingram School of Engineering
Texas State University
San Marcos, USA
aslan@txstate.edu

J. Jimenez
Ingram School of Engineering
Texas State University
San Marcos, USA
jj30@txstate.edu

***Abstract*— Performing modeling simulations with visualization and control can be rather difficult for dynamical models or other diverse applications. This work demonstrates the utility of the Julia programming language to solve this problem by being able to keep a simple condensed code and creating a powerful visual model with various controls. The simulation application is based on material handling tasks that are analyzed for human fatigue from repetitive operations. The aim is to detect human motions when performing the tasks and predict fatigue levels associated with these activities. The Julia simulation development addresses developing a computational solution to generate human subjects performing material handling operations synthetically. With the use of several Julia packages, the simulation development used the data from the Qualisys Motion Capture Systems and rendered each of the 41 points of data of a human subject performing lifting and lowering repetitive tasks.**

Keywords—Julia, Material Handling, Motion Capture, Simulation.

I. INTRODUCTION

Current simulation models lack the cohesion between low-level programming languages to perform faster calculations and high–level languages for visualization and interaction. This contributes to an increase in software complexity. Julia programming language [1] is designed for fast performance but has the syntax of high-level languages. Its ease and open-source approach make it one of the most accessible languages for simulation development.

The application for this model was motion analysis of fatigue from manual material handling operations. These manual operations range from lifting, pushing, pulling, carrying a load, or holding objects for a long time. These tasks are often performed repetitively throughout a given period, which causes the operator to experience fatigue; as fatigue increases, the operator's performance decreases. Without adequate resting time, the operators begin to overuse their bodies, increasing the probability of an injury.

The approach of this work is to develop a 3D-simulation environment using Julia. The data collected from the motion capture is done through the Qualisys software using twelve cameras. The Qualisys software helps the end-users visualize and inspect the data collection sections and provides tools to work with the captured data. The motions generate *xyz*-coordinate displacement information of each maker based on a pre-determined point of origin. The coordinate information is stored on a database in a Tab Separated Value (TSV) file format. The TSV file is then fed to the Julia-developed environment in where the motions are replayed frame-by-frame. The overall process described is shown in Fig. 1.

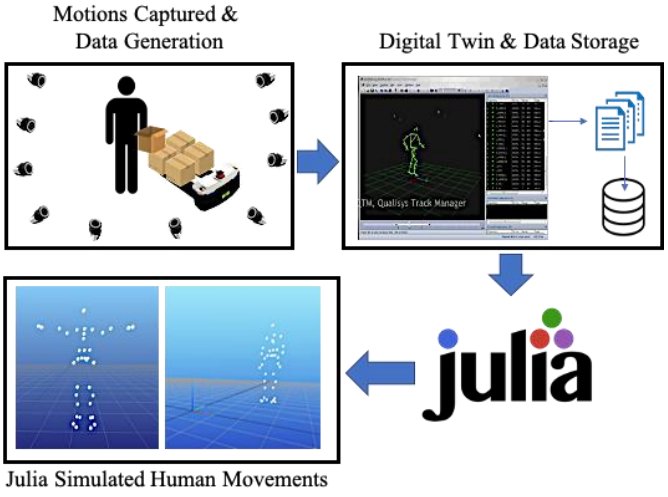

Fig. 1. Overall simulation development process using Julia

The Julia simulation recreated the digital twin representation of a human operator's motion-captured cameras performing lifting and lowering tasks. The Julia simulator was evaluated in time performance to compile all of the frames in the TSV file and render the markers in a 3D-space. This work's contribution impact is to expand the simulator to complement and generate more human subject motions when interacting with materials without needing human subjects. The simulator

will help researchers generate more synthetic data of the types of motions that need to be studied, the customization of the human operator body attributes, and the number of iterations that can be produced to indicate the capacity of repetitions a human can perform.

II. Background

A. Low-level and high-level programming languages

Low-level programming languages such as machine language or assembly language are much closer to something the hardware can understand. High-level programming languages such as C, C++, FORTRAN are closer to human-readable language and are mostly independent of what system they are used in. For applications where you need high-performance or are dealing with a specific architecture low-level language is the most optimized because it's very close to actual machine instructions which in return processes faster yielding results quicker. When needing to run applications across multiple platforms or want a user-friendly language, using a high-level language is optimal. This is because high-level languages enable users to write code fairly independently from any specific type of computer.

B. Modeling and Simulations

Modeling is the process of representing the environment that is experienced through interaction. A model can be a visual representation of the construction and working of some systems of interest. A simulation is a tool to evaluate a system's performance, existing or proposed, under different configurations of interest and over long periods of real-time [4]. In a computer aspect, simulations are created by running programs to realize a model. These programs can be small, running very quickly on any compatible device or large-scale that need extended periods to complete the task. The goal is to generate the material handling models through the coordinate captured values to generate a simulation of the experience human operator have during handling tasks.

C. Motion Capture

Motion capture was invented to speed up the animation process [5]. Motion capture is widely used today in gaming and the movie industry. Motion capture is defined as recording an object's movements, at high frequency, in a real-world setting by tracking the position of points of interest on the object [5][6]. There are various tracking technologies used for motion capture; they include acoustical, inertial, magnetic, mechanical, and optical systems [5][6]. The Acoustical systems use sound transmitters and microphones placed on specific locations to estimate the position of the points of interest. Inertial motion capture systems use an inertial sensor (containing an accelerometer and gyroscope) and rely on acceleration and angular velocity [6]. Magnetic based motion capture systems use a set of receptors to obtain the magnetic field given by the joint position, angles, and orientation on the body.

Mechanical tracking systems are the oldest and most straightforward method of capturing motion by using potentiometers to measure joint orientation displacement at each point. Optical marker tracking systems are divided into two categories: passive and active markers. The passive markers are reflective indicators attached to the point of interest on the actor. Passive marker positions are triangulated using the measurements captured from infrared high-speed camera reflections off the marker, producing 2D-coordinates of the data. Proprietary software is used to compute the 3D-coordinates of the markers [5]. Alternatively, active markers use LED markers that emit a light of their own instead of reflecting light. There are also optical marker-less tracking systems and use computer vision algorithms.

Fig. 2 show a pan-view of the motion capture lab with its surrounding cameras to capture the human subject's motions when interacting with the material. The data collection process is to have the human subject move between the two horizontal lines laid down on the floor to restrict the subject's movements during an interaction. The material is a box with a specified weight where the subject lifts it to a wooden table with a specified height. The subject will then lower the box after an interval of nine or fourteen seconds. The task of lifting and lowering is repeated until the human subject indicates the highest fatigue based on the Borg scale.

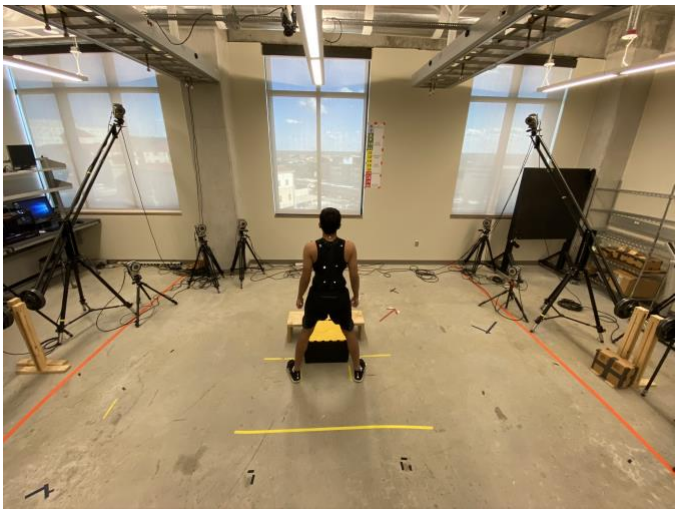

Fig. 2. Motion capture lab and configuration of human subject and cameras

D. Manual Material Handling

Manual material handling operations range from lifting, pushing, pulling, carrying with load, and holding objects for a duration of time. These manual operations are often performed repetitively throughout a task, causing the operator to experience fatigue. Once the operator experiences fatigue, their performance decreases. If they do not have adequate time to rest between tasks, they begin to overuse their body, causing injuries that may develop a work-related musculoskeletal disorder (WMSD).

One way to reduce the overuse of a worker's body will be to monitor the operator's movements using machine learning to detect fatigue. This solution will help reduce injuries or long-term effects from the manual material handling (MMH) operations. Prior work has use machine learning to measure fatigue in exercise motions using motion capture data or wearable sensors and a Rate of Perceived Exertion scale (RPE).

Using existing literature, a time-series solution using Recurrent Neural Network (RNN) based approaches known as Long-Short Term Memory (LSTM), and Gated-Recurrent Unit (GRU) will be considered for predicting the level of fatigue of a worker during a material handling operation [13].

A simulator for these types of tasks will help speed up the understanding of fatigue, motions, and kinematics of human operators. Machine learning techniques will help analyze the patterns that the simulator is generating over time and provide a human-analysis level of fatigue. Further investigation will be required to include possible machine learning inference modeling into the simulation, and Julia can support Julia packages' execution and communication with Python trained models.

III. Julia Packages

This section will give an overview of the packages used for visualization and calculations. Each package has Jupyter notebooks containing walkthrough examples and demos of their functionality.

A. Package: CoordinateTransformations.jl

The *CoordianteTranformations.jl* package manages simple or complex networks of coordinate system transformations [7]. These transformations can be easily manipulated concerning the input coordinates. The transformations are designed to be light-weight and efficient for real-time applications. Simultaneously, support for explicit and automatic differentiation makes it easy to perform optimization and, therefore, ideal for computer vision applications such as SLAM (simultaneous localization and mapping) [7]. The 41 markers' transformation is labeled with its *xyz*-coordinate values in millimeters from a reference point in the motion capture lab. The reference point is located in front of the human subject shown in Fig. 2 with a red arrow on the floor.

B. Package: MeshCat.jl

The *MeshCat.jl* visualizer package provides a 3D-interactive plane that allows the user to control camera control and the simulation model's visibility. *MeshCat.jl*is built on *three.js*, which is a JavaScript library and API that allows for the creation and display of 3D-animations in a browser [8]. The *MeshCat.jl* package is a successor to the previous visualizer, *DrakeVisualizer*. The interfaces are very similar, but the primary difference between the two is that *DrakeVisualizer* required Director, LCM, and VTK, while *MeshCat.jl* only needs a browser to execute. *MeshCat.jl* is designed to be lightweight and optimized for visualization in 3D-environments. The *MeshCat.jl* viewer runs entirely in the browser, with no external dependencies. All files are served locally, so no internet connection is required [9]. The 3D-environment places the marker information based on the origin point of the *MeshCat.jl* package. The simulation transformation of the raw data to Julia mimics the origin point placed in the motion capture lab.

C. Other Packages Used

- *GeometryTypes.jl*: Created a solid set of geometry types to be used in graphical/plotting applications. This package has since use been discontinued for *GeometryBasics.jl*. This package is provided to render the spherical shapes to indicate the 3D-position of each marker of a human subject. This can be changed with different parameter string values of desired geometrical shapes for the whole simulation or individual markers.
- *StaticArrays.jl*: Provides a framework for implementing statically sized arrays in Julia. In this instance, “statically sized” means that the type can determine the size, and “static” does not necessarily imply immutable [10]. This package provided the static environmental space that can be defined in the simulation regarding the origin point.

IV. Julia for Modeling and Simulations

A. Julia Programming Language

Julia is an open-source programming language distributed under the MIT license. Julia is the fastest modern open-source language for data science, machine learning, and scientific computing. Although the Julia version 1.0 was released in 2018, the language has been in development since 2009 by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. The goal for this team was to create a free language that was fast and easy to use. This project aims to leverage the high-level capacity of Julia’s APIs to generate a simulated human subject performing material handling tasks by using motion captured data.

B. Performing Simulations with Julia

Choosing a language to run simulations when dealing with the dependencies and libraries written in multiple languages can be a strenuous decision. In the simulation community, every modeling and simulation tool has its specialties, such as R is vital due to its statistics, data manipulation, and plotting libraries. MATLAB is known for its fast prototyping, compactness, and consistency. Julia wraps the best features of all these tools into its environment. Julia is as fast as C, dynamic as R, and interactive as MATLAB.

Although Julia is seemingly a compatible simulation high-level programming language, there are several reasons it is not yet widely used. First, Julia is a relatively new language in the programming world. There is very little documentation for this language, offering very little support when learning the language. Second, Julia is compiled at runtime. Languages such as C or C++ are compiled before execution. This results in longer first-time compilation times.

This paper aims to generate and render the digital twin aspect of the data into the Julia environment. Fig. 3 shows the digital twin's initial frame with its 41 markers of a human operator on the T-position. The *MeshCat.jl* generates the rendering package and added options in Fig. 3 provides ways to view different makers, environmental parameters, and animation options of the simulation. The options can help

visualize different perspectives, but the development of customized options will be investigated to generate different types of human operator parameters for different material handling tasks.

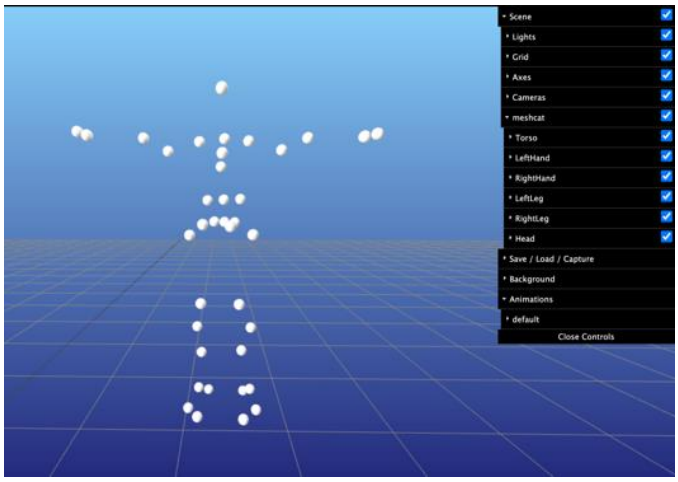

Fig. 3. Visualization of 41 markers on human operator using *MeshCat.jl*

V. Procedure

A. Data Collection

The actual collection process was performed in real-time using a collection of Qualisys twelve cameras used to track 41 reflective markers attached to the human subject. The cameras generate real-time *xyz*-coordinates as a function of time at 100 Hz (100 frames per second) while the subject operates. As the participant is performing the assigned movement, the team asked every minute their RPE using the Borg scale from six (No exertion) to twenty (Maximal exertion) [12][13] fatigue level experience during the task.

Once the data was captured using Qualisys, it was then exported into a TSV file with rows containing the *xyz*-coordinate for each frame and columns labeled for each marker. To collect the data for each frame, Julia could create a parsing function to obtain the correct *xyz*-coordinates for each marker. This created a two-element array containing an array for each coordinate in its specified frame. Once the data was correctly parsed into its correctly labeled variables, objects modeled by spheres were created in the *MeshCat.jl* environment.

B. Data Simulation

The development of the simulating started with the rendering of the human subject's upper torso, including the head, chest, shoulders, and arms. Fourteen data points in the preliminary simulation were used to test the flow of execution of the simulation. The time metric for the compilation and obtain a correct visualization of the points were capture to understand the impact of the number of frames a data collection can generate on a subject. The 41 maker points were then introduced to the simulation to complete the lower half of the subject. During this simulation, the compilation time required more memory space and processing execution time. The machine used for the development and simulation is a MacBook Pro with a 2.7 GHz Dual-Core Intel Core i5 and 8GB DDR3 RAM.

Each iteration followed the same procedure to keep the benchmarks consistent. The Juno Integrated Development Environment (IDE) was used for the development and configuration of Julia. For every iteration of the rendering simulation, a new Julia session was started in Juno to clear any cached information from the previous compilation. Table I show the total compilation timing of the three iterations launched with a different number of frames. The pattern of the timing values is exponentially incremental as the number of frames is doubled. The compilation time's concern can occur once the customization of different simulation parameters is included, and the frames.

Fig. 4 shows a rendering sequence of the simulation of the lowering and lifting tasks. The left-most image shows the T-pose in where the human subject is asked to spread their arm in such a pose as a marker if the starting point of the data collection. The other three images in the sequence show the maker *xyz*-coordinate information moving in its recorded displacement in its 3D-space. The sequence in Fig. 4 is the motion of lifting a weighted from the ground to a table at wrist-height.

TABLE I.

Total Compilation Time (s)					
Iterations	***6,000 frames***	***22,646 frames***	***45,293 frames***	***90,586 frames***	***181,172 frames***
1	56.214	65.742	100.762	294.465	1,640.876
2	53.025	63.883	182.928	425.343	1,350.137
3	55.355	71.726	151.910	462.952	1,529.061

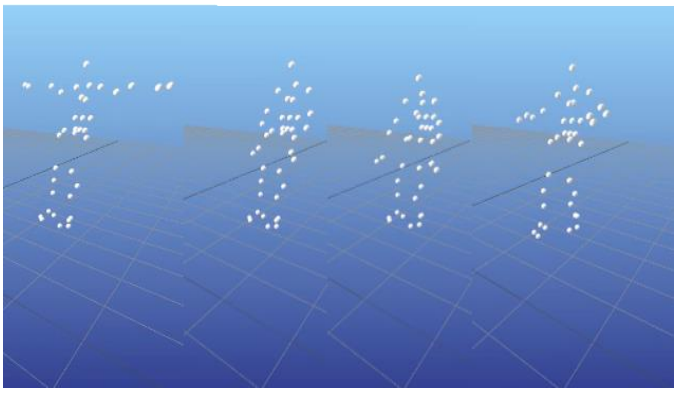

Fig. 4. Visualization of 41 markers on model using *MeshCat.jl*

VI. Conclusion

In current simulation environments, there exists a large dependency between low-level and high-level programming

languages. The Julia programming language provides a more straightforward solution to visualize and simulate small-scale models. The benchmarks demonstrate the feasibility of using packages for achieving a solution to replicate the digital twin data obtained from the motion capture cameras. The time analysis helps to better understand the type of computational resource required to develop and render the ideal simulations to generate experiments. This initial simulation effort has shown great promise to continue to develop utilizing Julia packages, render, and demonstrate the same actions that the Qualisys software package shows through its digital twin representation of the captured data.

VII. Future Work

Future work includes parallelization of the code to execute and compare processor performance and memory benchmarks. Currently, a serial compilation of a large dataset requires a vast number of resources. However, this will be an indicator of Julia's feasibility in large-scale simulation environments that support multi-processing HPC and GPU resources. Motion capture systems such as Qualisys are currently used to understand and capture fatigue levels of physical models. Julia's various packages and advancements in machine learning can help leverage a simulated model with interactive controls to produce faster results. The goal is to develop an overall simulation approach to synthetically simulate human subjects with different characteristics to material handling tasks in repetitive motions to analyze fatigue levels. The immediate impact is expected to reduce the timing of equipment configuration, calibration, subject preparation, and increase the number of human subject variations.

References

[1] Ivo Balbaert, Avik Sengupta, and Malcolm Sherrington. *JULIA: High Performance Programming*. Packt Publishing.

[2] MATLAB : a practical introduction to programming and problem solving / Stormy Attaway.

[3] Zhirkov, Igor. Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture. Apress, 2017.

[4] A. Maria, “Introduction to modeling and simulation,” in WSC ’97: Proceedings of the 29th conference on Winter simulation. Washington, DC, USA: IEEE Computer Society, 1997, pp. 7–13.

[5] P. Nogueira, "Motion capture fundamentals", A Critical and Comparative Analysis on Real-World Applications, pp. 1-12, 2011. [Accessed 28 October 2019].

[6] M. Field, Z. Pan, D. Stirling and F. Naghdy, "Human motion capture sensors and analysis in robotics", Industrial Robot: An International Journal, vol. 38, no. 2, pp. 163-171, 2011. Available: 10.1108/01439911111106372. [Accessed 28 October 2019].

[7] JuliaGeometry. “JuliaGeometry/CoordinateTransformations.jl.” *GitHub*, 21 May 2020, github.com/JuliaGeometry/CoordinateTransformations.jl.

[8] “Three.jsr117.” Three.js – JavaScript 3D Library, threejs.org/.

[9] Rdeits. “Rdeits/MeshCat.jl.” *GitHub*, 28 May 2020, github.com/rdeits/MeshCat.jl.

[10] JuliaArrays. “JuliaArrays/StaticArrays.jl.” *GitHub*, 23 May 2020, github.com/JuliaArrays/StaticArrays.jl.

[11] Qualisys. Motion Capture Systems [corporate website] Gothenburg, Sweden: Available from: http://www.qualisys.com. [Accessed 18 August 2019]

[12] G. Borg, "Psychophysical scaling with applications in physical work and the perception of exertion.", Scandinavian Journal of Work, Environment & Health, vol. 16, pp. 55-58, 1990. Available: 10.5271/sjweh.1815 [Accessed 18 December 2019].

[13] Hernandez, Geovanni, Valles, et. al. (2020). Machine Learning Techniques for Motion Analysis of Fatigue from Manual Material Handling Operations Using 3D Motion Capture Data. 10.1109/CCWC47524.2020.9031222.