Preprints, 10th IFAC International Symposium on
Advances in Automotive Control
Columbus, Ohio, USA, August 28-30, 2022

Co-simulation of the Unreal Engine and MATLAB/Simulink for Automated Grain Offloading

Chufan Jiang [*] Shveta Dhamankar [*] Ziping Liu [*]
Gautham Vinod [*] Gregory Shaver [*] John Evans [*]
Corwin Puryk [**] Eric Anderson [**] Daniel DeLaurentis [*]

[*] *Purdue University, West Lafayette, IN 47907 USA (e-mail: jiang420@purdue.edu).*
[**] *Deere & Company, Moline, IL 61265 USA*

Abstract: This paper presents a generic simulation platform with two widely employed software, Simulink and Unreal, to simultaneously simulate the perception in virtual 3D scenarios and system dynamics of automated systems. The proposed CoSim framework improves the accuracy and reduces the development time of automation systems for agricultural crop harvesting and transfer. Strategies using either cameras or LiDAR are supported by the framework. To demonstrate the capability of the proposed CoSim tool, this paper simulates an automated offloading process conducted by a combine-tractor system with a closed-loop controller and a LiDAR-based perception system. The simulation results show that CoSim can be used for both system design and system evaluation.

Keywords: Perception, simulation, driver assistance system, closed-loop control.

1. INTRODUCTION

An automated system usually has three key components: A perception system to observe the system states, a controller to regulate the system behavior, and the actuators to perform the control operations. For sophisticated automation tasks, an intelligent perception system is often desired to enable the extraction of contextual information to help coordinate system components. Simulation is a crucial step in the development and validation of the automated system. Given the key components of the automated system, a model to simulate both the controlled system dynamics and the perception system response would be beneficial to develop and validate the automated system.

However, it is challenging to simulate a perception-based autonomous system working in a dynamically changing environment. The system usually includes multiple objects, a multi-sensor perception system, and a control system. At the same time, it dynamically interfaces with the external environment by perceiving and changing the environment. Therefore, it is crucial to simulate how the perception module and control module impact each other, and how the whole system interacts with the environment through sensors and actuators.

Some simulation platforms designed for autonomous vehicle and robotics applications have proven successful. Gazebo Koenig and Howard (2004) is an open-source simulation platform that has been widely used in robotic research. It uses a modular design that supports different physics engines, robotics models, virtual sensors, and 3D world creation. Although Gazebo provides rich features and good compatibility, the features in its rendering engine are not as rich as Unreal or Unity, which makes it challenging to create a close-to-realistic visual environment. USARSim Carpin et al. (2007) is a high-fidelity robot simulator based on the Unreal Tournament game engine. USARSim provides built-in disaster environments, commercial and experimental robot models, and sensor models including camera and range finder. AirSim Shah et al. (2018) is an open-source, cross-platform simulator developed for drones and cars. The simulator builds on Unreal and provides comprehensive models for autonomous vehicle systems including vehicle model, environment model, physics engine, sensor model, and visual rendering. Flight control hardware with a decision-making engine can interact with the simulator to achieve real-time hardware-in-the-loop simulation.

All these platforms focused on specific automated objects (on-road vehicles or robots), and they simulated system dynamics in their own environment with a set of predefined models, which makes them less extendable to other applications, such as automated grain offloading that this paper targets at.

Therefore, more generic software is considered in this work to build a co-simulation platform that can be applied to different autonomous applications. MATLAB/Simulink is widely used by users in both the research and industry community. It provides various toolboxes to model complex mechanical systems, design sophisticated controllers, develop and implement advanced algorithms, as well as interface with other software and firmware for data transmission. The Unreal Engine is one of the most advanced creation tools for the gaming industry. It is open access and has been used to visualize a broad range of applications such as autonomous driving simulation, surgery simulation, maritime simulation, robotic simulation Resch et al. (2018); Leudet et al. (2019). Incorporating such popular and well-developed software enables users to transfer existing models to the CoSim platform more easily and to customize the project based on specific applications.

This paper proposes to a simulation platform with two widely employed software, Simulink and Unreal, so it can be suitable for various applications. The effort described in this paper is the first one the authors are aware of that demonstrated same-time co-simulation between MATLAB/Simulink and Unreal for automated systems. The proposed CoSim framework includes a virtual 3D world in Unreal and simulates system models and dynamics in MATLAB/Simulink. The feasibility of the CoSim tool is illustrated by an application on crop harvester grain offloading automation. This CoSim scheme can be transferred to other applications given the versatility of the software packages used.

2. METHODOLOGY

2.1 *Automatic offloading project*

The automatic offloading of grain from combine harvester to tractor driven grain cart is a motivating example to show how to use the proposed of the CoSim capability developed. The main purpose of automatic offloading is to automate grain unloading from a combine harvester to a tractor-driven grain cart while the combine is still moving and harvesting grain (e.g., corn, soybeans, or wheat). A perception system is used to monitor grain fill status and the grain flow impact location. Both cameras and LiDAR can be simulated in the CoSim and data processing algorithms based on LiDAR sensor are developed to extract the key perception information for the controller. The control system determines the designed auger location based on fill strategy and perceived grain profile, and controls the vehicle location and auger status to achieve the desired fill level without grain spillage from the grain cart.

2.2 *Simulation architecture*

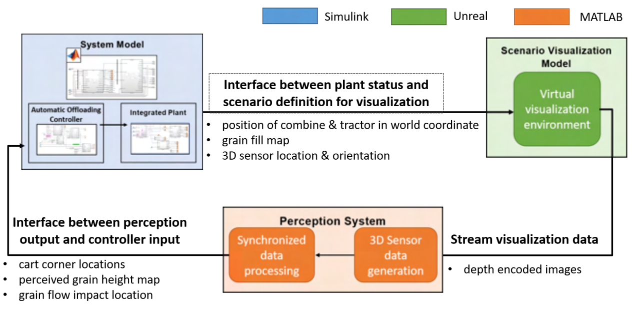

Simulink
Unreal
MATLAB
System Model
Automatic Offloading Controller
Integrated Plant
Interface between plant status and scenario definition for visualization
• position of combine & tractor in world coordinate
• grain fill map
• 3D sensor location & orientation
Scenario Visualization Model
Virtual visualization environment
Interface between perception output and controller input
• cart corner locations
• perceived grain height map
• grain flow impact location
Perception System
Synchronized data processing
3D Sensor data generation
Stream visualization data
• depth encoded images

Fig. 1. Framework of co-simulation platform

As shown in Fig. 1, the co-simulation for perception in the loop automation architecture includes three major model modules. First, the system model simulates the behavior of the automatic offloading controller and the motion dynamics of the combine harvester and a tractor-driven grain cart, as well as the grain piling dynamics inside the cart. Second, the scenario visualization model creates a photorealistic virtual unloading environment in response to the system model status, including the position of the vehicles, grain fill level, and the placement of the perception sensor(s). Third, the perception system then models the 3D LiDAR sensor using scenario visualization data and runs a data processing algorithm synchronously to extract useful perception information for unloading status monitoring (e.g., cart corner location, perceived grain height map). The output from the Perception System Module is then sent back to the controller in the System Model Module, so the controller can estimate the current unloading progress and how full the cart is, then send commands to vehicles to reach a certain fill target while avoiding spillage.

The color code in the architecture diagram labels the development environment for each module. The blue box in Fig. 1 is developed and run in Simulink, the green box is developed and run in Unreal and the orange blocks are developed and run in MATLAB.

2.3 *System Model Module*

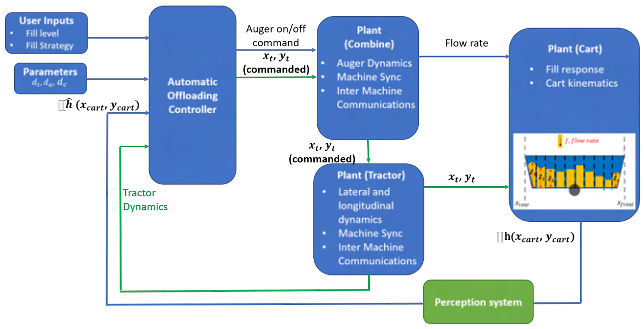


Fig. 2. System model architecture

The blue blocks in Fig. 2 depict the components in the system model and its relation to the visualization model and perception system shown in the green block. The system model can be divided into two parts: the automatic offloading controller and the vehicle models.

During automatic offloading, the operator sets a target fill level with the desired fill strategy. Based on the user input, pre-known parameters (e.g., vehicle geometry), and feedback from the vehicles and the perception system, the automatic offloading controller calculates the control efforts: auger on/off command and vehicle location command. Machine Sync, a commercially available technology by John Deere is used to control the relative position between the two vehicles. The vehicle models simulate how tractor and combine dynamics are affected through use of Machine Sync. On top of that,the modeled auger flow rate and the vehicle relative location are used together with the known cart geometry to run a grain fill model which simulates how the grain profile progresses inside the grain cart. Finally, the grain profile and other system status are sent to Unreal for visualization. The grain profile is monitored by the perception system and sent back to the automatic offloading controller, closing the automatic offloading control loop.

2.4 Scenario Visualization Model Module

The main purpose of the scenario visualization model is to create a 3D world that is consistent with the system status simulated by the system model, and then generate synthesized camera images from the current scenario. It also generates depth encoded images that will be later used to simulate LiDAR data.

The 3D virtual scenario is defined by vehicle status, grain property, environment lighting conditions and camera configuration. The vehicles CAD models are loaded into Unreal to achieve a truthful representation of vehicle geometries as shown in Fig. 3a. The vehicle dynamics from the system model is used to dictate the vehicle movement in 3D visualization. A grain flow element is added in the 3D scenario using the Unreal particle emitter, and a 2D grain fill map simulated by the grain fill model is converted to a 3D grain mesh to reflect the grain profile change during unloading. The built-in camera model in Unreal is used to simulate 2D camera images with customizable configurations including placement and camera parameters including the field of view, aspect ratio and projection mode. Figure 3b shows an example of a camera sensor capturing an unloading process with grain flow and grain bed visualization.

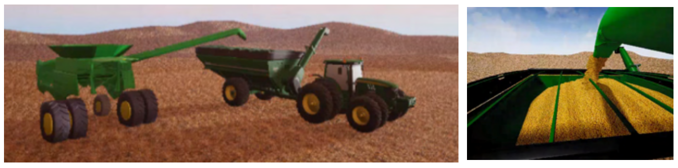

(a) (b)

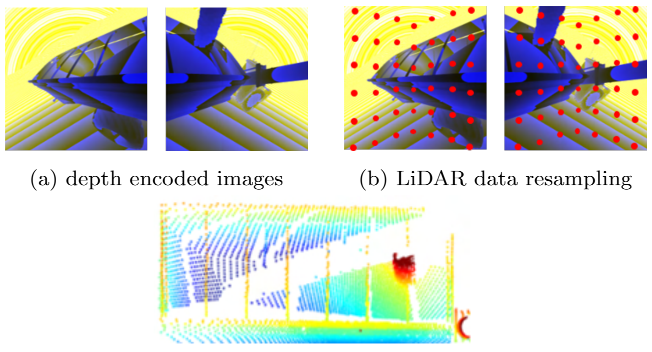
Fig. 3. Example elements in scenario visualization model: (a) combine and tractor models; (b) camera image from an unloading process.

2.5 Perception System Module

In the perception system module, the authors built a LiDAR sensor model based on the Unreal camera model. The simulated sensor data is then sent to data processing algorithms to extract useful context information.

LiDAR sensor model The proposed LiDAR sensor model is developed in three steps. First, the depth information was encoded together with the camera RGB image. Second, the 3D coordinate for each camera pixel was then subsequently retrieved from the depth image. Third, since the LiDAR sensor and camera sensor sample the world in different patterns, a data resampling was required to generate point cloud data following the LiDAR pattern. The first step was coded in the Unreal with its visual scripting system, and the following steps were programmed in MATLAB/Simulink function blocks.

Step1: Render camera image with depth information (in Unreal). A LiDAR sensor provides a 3D measurement of the environment geometry. However, the virtual camera in Unreal only provides a 2D RGB image, and does not directly provide depth information for the camera sensor. To simulate a 3D LiDAR sensor, 3D information from the sensor point of view is required, and therefore


(a) depth encoded images (b) LiDAR data resampling

(c) simulated LiDAR point cloud

Fig. 4. LiDAR data simulation

must be calculated. Although Unreal does not directly provide a 3D sensor model, it allows users to retrieve the depth information for each pixel in the rendered image. The authors developed a pixel shader to encode the depth information of each pixel into color values (as shown in Fig. 4a).

In Unreal the depth information is a float number and the color image is stored by an 8-bit integer for each R, G, B channel. To encode the depth value into color image with a broad range and high precision, the following equations are used for depth encoding:

$$R(u,v) = G(u,v) = round(\min((\frac{d(u,v)}{\alpha})^{\gamma}, 1) \times 255) \quad (1)$$

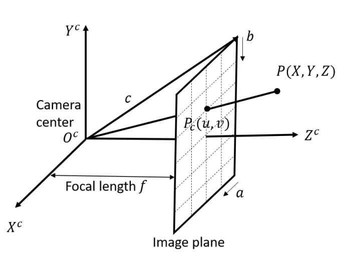
$$B(u,v) = round((\frac{d(u,v) \mod \beta}{\beta})^{\gamma} \times 255) \quad (2)$$

where $R(u,v)$, $G(u,v)$, $B(u,v)$ are red, green and blue channel of pixel (u,v) on the depth encoded color image. Each channel is stored by an 8-bit integer so the value is within $[0,255]$. $d(u,v)$ is the depth value to be encoded with the unit of *cm*, α and β are constant scaling factors with $\alpha > \beta$. α should be larger than the maximum value of d to avoid data truncation. The *mod* is the modulo function and *round* is the rounding function. Unreal internally applies an inverse gamma correction on texture color for better visualization. Therefore, a gamma function was added in the depth encoding with the gamma correction constant $\gamma = 2.2$.

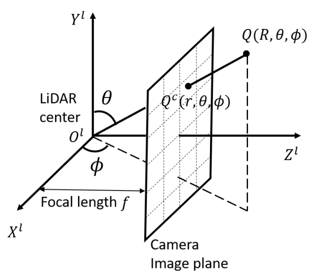
Step2: Retrieve 3D coordinate from depth image (in Simulink). After generating the depth encoded image, the 3D coordinate for each pixel (u,v) on the image can be retrieved. Since the depth value is encoded into R, G, B channels of a color image, the decoding can be achieved in an inverse manner using Eq. 3

$$d'(u,v) = \left[round(\frac{R(u,v)}{255} \times \frac{\alpha}{\beta} - \frac{B(u,v)}{255}) + \frac{B(u,v)}{255}\right] \times \beta, \quad (3)$$

where $d'(u,v)$ is the decoded depth value at pixel (u,v). The depth $d'(u,v)$ is the Z coordinate on camera sensor coordinate. To retrieve 3D information from a 2D depth image, the camera model used in Unreal needs to be investigated.



(a)



(b)

Fig. 5. Sensor models: (a) the pinhole camera model; (b) the LiDAR model built upon the camera model.

Unreal uses a basic pinhole model Hartley and Zisserman (2003) as shown in Fig. 5a to simulate how the camera sensor captures an image of the 3D world. O^c is the camera center and $X^cY^cZ^c$ is the camera 3D coordinate. The image plane is perpendicular to the Z^c axis with a constant distance f called focal length and Z^c passes through the center of the image plane. A 3D point $P(X,Y,Z)$ is mapped to a 2D point $P_c(u,v)$ on the camera image where $P_c(u,v)$ is the intersection between the line O^cP and the image plane. Equation 4 describes the mapping relationship from a 3D point to a 2D image pixel

$$\overrightarrow{O^CP} = (\bar{a}u + \bar{b}v + \bar{c})W \tag{4}$$

Since $P = [X,Y,Z]^T$ and $O^C = [0,0,0]^T$, Eq. 4 can also be written as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \bar{a} & \bar{b} & \bar{c} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} W. \tag{5}$$

Here $\bar{a}$ and $\bar{b}$ are unit vectors of two axes of the image plane with $\bar{a} = [1,0,0]^T$ and $\bar{b} = [0,-1,0]^T$. $\bar{c}$ is the vector from camera center to the origin of the image plane. For a camera sensor with $w \times h$ resolution and $hfov$ degrees horizontal field of view, $\bar{c} = \left[-\frac{w}{2}, \frac{h}{2}, \frac{w}{2\tan(hfov/2)}\right]^T$. W is the augmented dimension in homogeneous coordinate that is used for simplifying the calculation. In Eq. 5, the coordinate of each camera pixel (u,v) is known, and the decoded depth $d'(u,v)$ from Eq. 3 is the Z coordinate of P. Therefore, there are three unknowns X, Y, W and three equations in Eq. 5, the 3D coordinate $P(X,Y,Z)$ can be solved for each pixel $P_c(u,v)$ of the depth image.

Step3: Resample camera image based on LiDAR patterns (in Simulink).

Camera and the LiDAR sensors sample the world in different ways. The camera pixels on the image plane distribute evenly in horizontal and vertical directions. While the LiDAR sensor samples the world evenly in angle instead of in distance. After getting the depth encoded image from Step 1, the image is resampled based on LiDAR's scanning patterns. As illustrated by Fig. 5b, the LiDAR sensor scans the world with evenly spaced angles and thus its point cloud data is typically represented by spherical coordinates with a distance R, an azimuth angle θ, and a polar angle ϕ. The goal is to derive the LiDAR model from the camera model and thus LiDAR coordinate $X^lY^lZ^l$ is aligned with the camera coordinate. Then the image point $Q^c(r,\theta,\phi)$ corresponding to Q can be determined on the image plane. Since Q^c and Q lie on the same LiDAR ray, they share the same angles θ and ϕ. However, Q^c has to be on the image plane and it has a different distance from Q. Consider the fact that the camera image plane is perpendicular to Z-axis with a constant distance of focal length, the distance r can be computed based on the focal length and two angles:

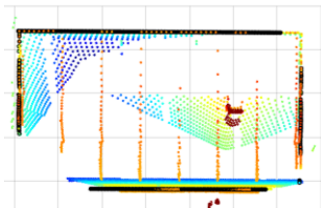
$$r = f/\sin(\theta)/\sin(\phi). \tag{6}$$

Here $f = \frac{w}{2\tan(hfov/2)}$, θ is integer times of LiDAR vertical resolution and ϕ is integer times of horizontal resolution. Now the spherical coordinate of Q^c can be converted into the camera image pixel (u^l, v^l) by Eq. 7.

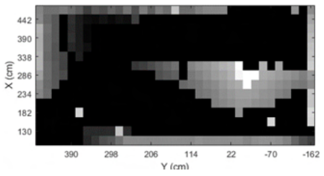
$$\begin{aligned} u^l &= round(r\sin(\theta)\sin(\phi) + w/2) \\ v^l &= round(r\cos(\theta) + h/2) \end{aligned} \tag{7}$$

Through Eqs. 6 and 7, each LiDAR sample point is mapped onto one camera pixel (u^l, v^l) and then extract the 3D coordinate of this pixel from depth image using Eq. 5 in Step 2. Typically the LiDAR sensor has a larger horizontal field of view than what a camera sensor has. Therefore, two camera models with the same location but different viewing angles are stitched together to simulate one LiDAR sensor with an ultra-wide field of view.

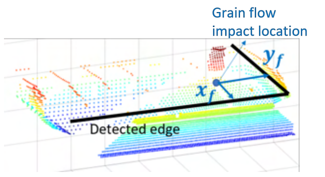
(a) Grain cart picture from the sensor view



(b) Cart boundary detection



(c) Grain height map



(d) Grain flow impact location

Fig. 6. Data processing algorithms for automatic offloading

Data processing algorithm To achieve closed-loop simulation with perception in the loop, the perception system needs to perceive the contextual information from simulated sensor data and send it to a controller for decision making. Therefore, data processing algorithms are implemented in the Perception System Module.

For the automatic offloading project, the perception system should provide three key pieces of information to the controller: the location of the grain cart boundary, the grain height estimation of the current grain pile in the cart, and the impact location within the cart. Figure 6a is a picture of the perceived grain cart from the sensor view. Figure 6b is an example of cart boundary detection results on simulated LiDAR data where the black dots labels the detected boundary data. With the cart boundary location, the grain pile profile is discretized into a 2D height map in Fig. 6c. Each grid from the map represents the grain height relative to the cart boundary. The gray-scale color code reflects the height value and black grids are invalid measurement. Besides, the perception system also detects grain flow impact location which is the contacting point between grain flow and the grain bed, ash shown by the blue dot in Fig. 6d.

Note that the data processing algorithms described above can be replaced by any other customized algorithms, and users can always take advantage of the powerful MATLAB toolboxes for more advanced algorithm development.

2.6 Data transmission interface

As shown in Fig. 1, simulation data are transmitted across Unreal and MATLAB/Simulink to achieve communication between different modules of the CoSim framework. There are three types of data exchanged between Unreal and MATLAB/Simulink: floating-point number and camera data.

The bi-directional numeric data exchange is facilitated by a shared memory interface developed by MathWorks Jayaraman et al. (2017). In Unreal, the interface provides a plugin to enable necessary read/write functions, and in MATLAB it includes a block for reading and writing data from the same shared memory locations specified in Unreal Engine, MATLAB, and Simulink.

For camera data transmission, Mathworks provides an interface plug-in in Vehicle Dynamics Blockset MathWorks (2020) to enable configuration of Unreal virtual camera from MATLAB/Simulink, including loaction, orientation, and specifications of the camera. To establish image transmission in Unreal, a Sim3dSceneCap actor with a unique tag name is added to the virtual scene.

3. SIMULATION RESULTS

3.1 Offloading simulation example

The developed CoSim framework was used to simulate an automatic offloading process with the LiDAR-based perception feedback and closed-loop control.

The automatic offloading system includes a combine harvester (model: John Deere S660), tractor (model: John Deere 8345R IVT), and grain cart (model: Brandt 1020XR). The perception system is configured assuming a LiDAR sensor (model: Waymo Honeycomb) with data processing algorithms developed by the authors. The unloading is on flat terrain and the vehicles are moving in straight lines. The unloading was finished in one continuous load. Visualization 1 shows this process. The simulation in the video runs a Front-to-Back control strategy to fill up the grain cart from a half-full status to the desired fill level.

Figure 7 shows multiple windows that pop-out during CoSim and each of them represents the simulation result from different parts of the closed-loop system. The top left figure shows the combine-tractor relative position from vehicle position control. The top right figure is a 2D scheme of how the grain profile looks like. This system status is sent into the Scenario Visualization Module to allow updates to the 3D automatic offloading environment. The Unreal window on the right shows an example of the relative position between the combine auger and grain cart in a top-down view. The perception system in the orange

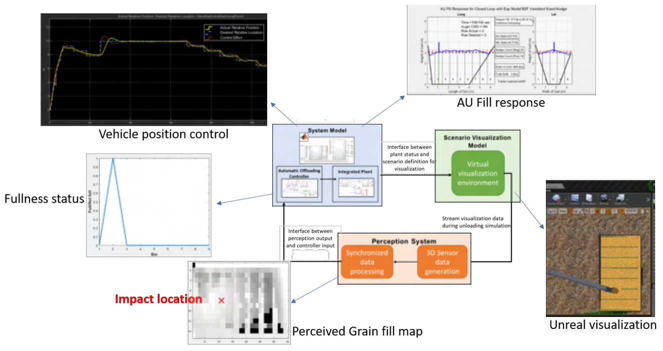
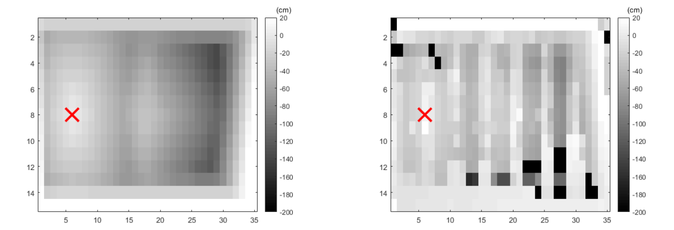

Vehicle position control
AU Fill response
Fullness status
System Model
Automatic Offloading Controller
Integrated Plant
Interface between plant status and scenario definition for visualization
Scenario Visualization Model
Virtual visualization environment
Stream visualization data during unloading simulation
Interface between perception output and controller input
Perception System
Synchronized data processing
3D Sensor data generation
Impact location
Perceived Grain fill map
Unreal visualization

Fig. 7. Simulation result windows in CoSim platform



(a) Ground truth grain fill map from the grain fill model

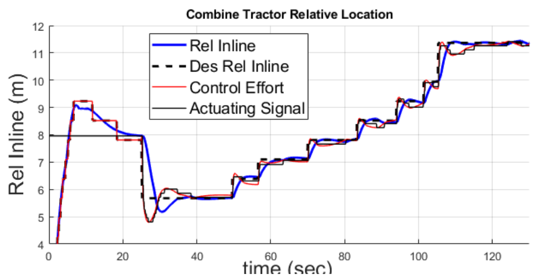
(b) Grain height map generated by perception system

Fig. 8. Comparison between the perceived grain fill map and ground truth

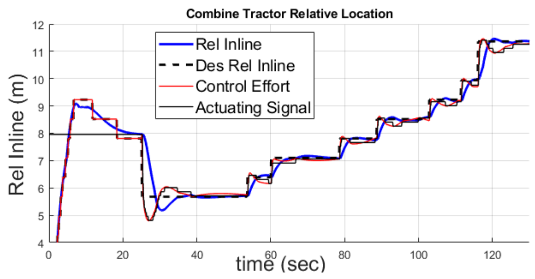
box inputs 3D sensor data from the Unreal environment and feeds the data into the perception algorithm. The example perception result is shown in the bottom left with an impact location on top of a 2D grain fill map. Based on this grain fill map, the controller inside System Model Module (blue box) estimates the fullness status for each grain cart section as shown on the left fullness status plot where 1 means full and 0 means not full. The fullness status drives decision making inside the automatic offloading controller. Once the current unloading section is full, the controller will command the auger to the new unloading position. In this CoSim simulation framework, each part of the co-sim block is interacting with others as a loop and they progress in a complex dynamic to generate final unloading results. Visualization 1 is an example of the CoSim demonstrating that in the simulation environment the designed automation system achieves unloading the grain from combine to grain cart successfully.

3.2 Automatic offloading evaluation

The unloading results with perception model feedback and ground truth feedback were compared to quantify the influence of the perception model error. Figure 8 shows the grain fill map from ground truth information generated by the grain fill model and the perception system. Each grid has a gray-scale color representing the averaged grain height within it and the grid without a valid measurement is colored as purely black. The grain fill model provides the ground truth of grain fill level because it defines the visualization scenario from which the sensing data is generated. As shown in Fig. 8b, the grain map from the perception system is incomplete and noisy. The CoSim can be used as a tool to quantify the impact of perception system error.

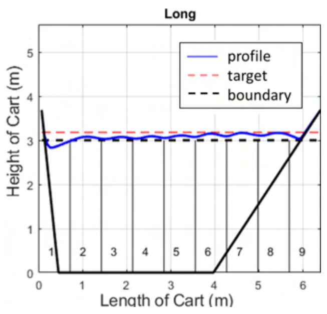


(a) Use ground truth as grain fill level feedback for controller

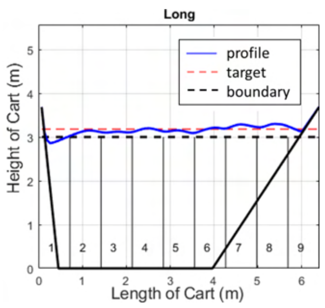


(b) Use perception system result as grain fill level feedback for controller

Fig. 9. Vehicle relative location plots



(a)



(b)

Fig. 10. Grain fill status plots: (a)Use ground truth as grain fill level feedback for controller; (b) Use perception system result as grain fill level feedback for controller.

As an example, Fig. 9 compares the combine auger location relative to the grain cart during an automatic offloading process with ground-truth feedback (Fig. 9a) and perception system feedback(Fig. 9b). The black dash line is the desired auger location based on the cart fullness feedback. The grain cart is divided into 9 sections in the longitudinal direction. Once the current unloading section is full, the desired auger location moves to the next one. The red solid line is the control effort calculated by the controller and the black line is the actuating signal with multiple nudge command. The blue line shows the actual auger location. In the perception system feedback result shown in Fig. 9b, due to the incomplete and noisy perception results, the controller cannot always get an accurate and up-to-date measurement on the grain profile. As a result, the desired location from the controller changes less smoothly. For example, the auger location changes more frequently around 60 sec and stay longer on the current section between 60 - 80 sec.

This is also reflected on the final grain profile after unloading finishes. Figure 10 shows a cross-section view of the final unloading profile in the longitudinal direction. The blue line is the grain profile, the black line indicates the height level of the cart boundary, and the red dash line marks the target fill level.It is desirable to have a final profile that is in between the red and the black line. With the ideal feedback of grain fill status each grain peak is right on the target fill level (Fig. 10a). On the contrary, the unloading process with the imperfect perception results generates a grain pile that is not uniform and exceeds the target level shown with the red dashed line (Fig. 10b).

4. SUMMARY

This paper presents a co-simulation (CoSim) framework with Unreal and MATLAB/Simulink and its use case to simulate automation systems with perception feedback. The developed framework can simulate both the system dynamics in Simulink and perception system dynamics in Unreal synchronously. Simulation results on an automatic offloading system demonstrates that the presented CoSim can be used to develop and validate an automated system and provides tools for system evaluation and optimization.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge Deere and Company for providing financial and technical support for the project. We would also like to thank MathWorks for the software support.

REFERENCES

Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). Usarsim: a robot simulator for research and education. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 1400–1405. IEEE.

Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision.* Cambridge University Press.

Jayaraman, A., Micks, A., and Gross, E. (2017). Creating 3d virtual driving environments for simulation-aided development of autonomous driving and active safety. Technical report, SAE Technical Paper.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, 2149–2154. IEEE.

Leudet, J., Christophe, F., Mikkonen, T., and Männistö, T. (2019). Ailivesim: An extensible virtual environment for training autonomous vehicles. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, 479–488. IEEE.

MathWorks (2020). Vehicle dynamics blockset. `https://www.mathworks.com/products/vehicle-dynamics.html`. Accessed: 2020-12-15.

Resch, J., Ehrentraut, J., Barnett-Cowan, M., et al. (2018). Gamified automation in immersive media for education and research. *arXiv preprint arXiv:1901.00729.*

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 621–635. Springer.