Tractor-Implement Real Time Interactive 3D Simulation Based on openFrameworks and OpenGL

Benjamin Fernandez¹, José Antonio Cerrada¹, Joseph Gross²

¹Departamento de Ingeniería de Software y Sistemas Informáticos, UNED, C/ Juan del Rosal, Madrid, España
²Institut für Zuverlässigkeitsotechnik, TUHH, Hamburg, Germany

ffernande1111@alumno.uned.es
jcerrada@issi.uned.es
joseph.gross@tuhh.de

Keywords: Tractor-Implement Simulation, OpenGL, openFrameworks, Tractor Guidance Simulation.

Abstract: This article presents a Tractor-Implement Simulation using C++, openFrameworks and OpenGL. It provides 3D animation and a friendly user interface for online interaction. This facilitates understanding, learning, design and control tuning of off-road guidance and navigation systems.

1 INTRODUCTION

Current multidisciplinary tendencies in which artists and engineers work together have led to the development of tools such as openFrameworks that not only help the artist to express him- or herself in a digital realm, but allows the engineer to use powerful audiovisual aids in an easy way that facilitates learning as well as the implementation of interactive real time systems.

The process of setting up an autonomous off-road vehicle is very complex and the environment plays a very important role. Especially if the navigation system is based on a satellite positioning system, one would need enough space and a good connection to the satellite – not to mention the weather conditions. For instance, to set up a GNSS (Global Navigation Satellite System) system in a tractor, one will need, apart from the tractor being able to accept steering commands, a lot of resources including the receiver, an Electronic Control Unit, a Real Time Kinematics Station for correcting the signal and finally a modem to connect to it. Then, the system has to be tuned for different conditions and with different implements for the expected range of velocities. Moving all these resources around requires substantial effort and coordination. Only the tuning process could take up to 5 days and this has to be repeated for each model of a vehicle. Furthermore, in the development phase, the software as well as the hardware could change, which would require retuning the whole system. Also, changes in the ground or soil will change the performance of the lane-tracking system and ideally one should consider and test the different changes in the environment to ensure a robust and reliable guidance system.

One solution for saving time and resources could be to simulate the system. Nevertheless, simulating nonlinearities is not an easy task. There are different tools for simulating multi-body dynamics such as MATLAB, ADAMS CAR and SIMPACK among many others. The method in this paper is aimed at simulating an interactive tractor-implement in a more straight-forward, compact manner using openFrameworks, a C++ toolkit.

2 METHODS

openFrameworks is a C++ toolkit that collects well-known and powerful libraries, such as OpenGL, OpenCV, GLU and Poco among others, for processing graphics, audio, fonts, video, computer vision and 2D and 3D rendering. It works on a multi-thread level and offers encapsulation of complexity for programing simple applications on a higher level (Nimoy, 2016). It is available for Windows, Linux, OSX, iOS and Android. It supports also the platforms ARMv6 and ARMv7 under Linux (e.g. Raspberry-Pi).

The architecture used for the simulation is model-based and it is presented in the first part of
in this chapter together with the control theory, whereas the second part presents the C++ implementation with openFrameworks which allows real time interaction and 3D rendering.

2.1 Block Diagram and System Architecture

From a control systems perspective, the base representation of the system was done with three blocks \( (G_{pr}, G_{pr}, G_{pr}) \) for the steering system, angular velocity and lateral position, respectively, allowing the implementation of a cascade controller (Derrick and Bevly, 2009). For the sake of the simulation, the steering system \( G_{pr} \) is already in a closed loop (plant with a feedback controller). The lateral position controller is represented by \( G_{cr} \) and the yaw rate controller by \( G_{crf} \) & \( G_{crff} \). Each block of Fig. 1 was simulated as a C++ object (except from \( G_{cr} \) and \( G_{crff} \) which were packed into one object) and runs as an independent task making a total of five C++ objects.

![Figure 1: Navigation system represented in a Block Diagram.](image)

2.1.1 Tractor-Implement Model and Controller (Yaw Rate)

The tractor model \( (G_{pr}) \) from the block diagram of Figure 1, is based on a bicycle model (Fig. 2).

![Figure 2: Bicycle model of Tractor with an implement (Hitch).](image)

The mathematical representation can be derived using Newton’s second law of motion. According to Gillespie, 1992 and (Derrick and Bevly, 2009) the lateral forces (front rear and hitch) are a function of the slip angles where the proportionality is defined as the cornering stiffness. Therefore substituting Eq. (1) in the equation of motion

\[
\begin{align*}
F_f &= -c_{af} \alpha_f \\
F_r &= -c_{ar} \alpha_r \\
F_h &= -c_{ah} \alpha_h
\end{align*}
\]

we get the following state space model (Pearson and Bevly, 2008):

\[
\begin{bmatrix}
\dot{y} \\
\dot{r}
\end{bmatrix} =
\begin{bmatrix}
a_{00} & a_{01} \\
a_{10} & a_{11}
\end{bmatrix}
\begin{bmatrix}
y \\
r
\end{bmatrix} +
\begin{bmatrix}
b_{00} \\
b_{10}
\end{bmatrix} \delta
\]

where

\[
\begin{align*}
a_{00} &= -c_{ah} + c_{ar} + c_{af} \\
a_{01} &= (b + c) c_{ah} + b c_{ar} - a c_{af} \\
a_{10} &= (b + c) c_{ar} + a c_{af} \\
a_{11} &= \frac{(b + c)^2 c_{ah} + b^2 c_{ar} + a^2 c_{af}}{l_2 v_x} \\
b_{00} &= \frac{c_{af}}{m} \\
b_{10} &= \frac{c_{af}}{l_2}
\end{align*}
\]

For the simulation with openFrameworks, the parameters of the state space system were substituted with those of a Fendt tractor 939 shown in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1.6965 [m]</td>
</tr>
<tr>
<td>b</td>
<td>3.9585 [m]</td>
</tr>
<tr>
<td>c</td>
<td>2.1900 [m]</td>
</tr>
<tr>
<td>axis width</td>
<td>2.1510 [m]</td>
</tr>
<tr>
<td>m</td>
<td>18500 [Kg]</td>
</tr>
<tr>
<td>( l_{xz} )</td>
<td>2000 [N/deg]</td>
</tr>
<tr>
<td>( C_{af} )</td>
<td>4000 [N/deg]</td>
</tr>
<tr>
<td>( C_{ar} )</td>
<td>1600 [N/deg]</td>
</tr>
<tr>
<td>( C_{ah} )</td>
<td>0 to 6000 [N/deg]</td>
</tr>
<tr>
<td>( V_x )</td>
<td>2 to 20 [m/s]</td>
</tr>
</tbody>
</table>

For the simplification of the simulation, the input signal used is curvature \([Km^{-1}]\) instead of steering angle \(\delta\):

\[
\text{curvature} = 1000/\text{radius}[m] \\
\text{radius} = \frac{\text{axisWidth}[m]}{\sin(\delta)}
\]
Therefore, a modified version of the controller found in (Derrick and Bevly, 2009) for the yaw rate (feedback and a feedforward) was used to get Equations (5) and (6).

\[
G_{Cr} = \frac{cur_{fb}(s)}{r_{err}(s)} = k_{fb} \\
cur_{fb}(t) = k_{fb} \cdot r_{err}(t)
\]  

(5)

and

\[
G_{Crff} = \frac{cur_{ff}(s)}{r_{des}(s)} = k_{ff} \\
cur_{ff}(t) = k_{ff} \cdot r_{des}(t)
\]  

(6)

Where \(cur\) represents the curvature and the desired curvature is then the sum of the feedback-curvature and the feedforward-curvature as represented in Eq. (7).

\[
cur = cur_{fb}(s) + cur_{ff}(s)
\]  

(7)

2.1.2 Steering Model and Controller

For the simulation of the steering system, an identification of the closed loop of the hydraulic steering system of a Fendt 939 was performed. The identification algorithm used was ARX (Auto Regressive model with eXternal input). The identified digital transfer function was of fourth order with a sampling time of 20 milliseconds and its digital state space representation is shown in Eq. (8).

\[
x(t + Ts) = \begin{bmatrix} 1.8184 & 1 & 0 & 0 \\ -1.1826 & 0 & 1 & 0 \\ 0.65624 & 0 & 0 & 1 \\ -0.30751 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} -0.000859 \\ -0.014128 \\ 0.028073 \\ 0.00000 \end{bmatrix} u(t) + \begin{bmatrix} 1.8184 \\ -1.1826 \\ 0.65624 \\ -0.30751 \end{bmatrix} \cdot x(t - Ts)
\]  

(8)

\[
y(t) = [1 \ 0 \ 0 \ 0] \cdot x(t)
\]

2.1.3 Lateral Position and Controller

Knowing the forward and lateral velocities and the yaw rate \(V_x, V_y, \phi\) in body coordinates at time instant \(t\), one can calculate the position and heading in body coordinates at time \((t + 1)\) in relation to the previous measurement \(t\) after a time interval has elapsed as shown in Eq. (9) (Rovira Más, 2011).

\[
x_{b,t+1} = x_b \cdot \Delta t \\
y_{b,t+1} = y_b \cdot \Delta t \\
\psi_{t+1} = \psi \cdot \Delta t
\]  

(9)

Furthermore, knowing the current position in site coordinates and the heading angle \(\phi\), one can calculate the expected position in site coordinates at time \((t + 1)\) with Equation (10).

\[
\begin{bmatrix} x_{s,t+1} \\ y_{s,t+1} \end{bmatrix} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} x_{b,t+1} \\ y_{b,t+1} \end{bmatrix} + \begin{bmatrix} x_s \\ y_s \end{bmatrix}
\]  

(10)

Substituting Eq. (9) into (10) we obtain:

\[
\begin{bmatrix} x_{s,t+1} \\ y_{s,t+1} \end{bmatrix} = \begin{bmatrix} \cos(\psi \cdot \Delta t) & -\sin(\psi \cdot \Delta t) \\ \sin(\psi \cdot \Delta t) & \cos(\psi \cdot \Delta t) \end{bmatrix} \cdot \begin{bmatrix} V_x \cdot \Delta t \\ V_y \cdot \Delta t \end{bmatrix} + \begin{bmatrix} x_s \\ y_s \end{bmatrix}
\]  

(11)

The control law used for the lateral position is a PID (Proportional Derivative Integral) of the form:

\[
G_{Cl}(s) = \frac{r_{des}(s)}{i_{err}(s)} = k_{pl} \cdot \frac{1}{s} + k_{dl} \cdot s
\]

\[
r_{des}(t) = k_{pl} \cdot i_{err}(t) + \frac{1}{T} \int_{t_0}^{t} i_{err}(\tau) d\tau + T \cdot i_{err}(t)
\]  

(12)

2.2 Implementation with openFrameworks

As one can see in Figures 3 and 4, the application consists of 7 Classes or Objects. Two of them (OglApp and GuiApp) create a window each for the graphical representation where GuiApp is considered the main thread. It connects the listeners and events of the different parts of the simulation as well as renders a 2D user interface for the interaction with the system. It is capable of generating plots as well for a better understanding of what is happening with the different signals such as curvature, heading etc. OglApp is a thread as well and creates a window that exclusively gets the actual position and renders a 3D Tractor model with the use of OpenGL. The other five classes construct the code for the mathematical representation of each part of the block diagram (Fig. 1) of the tractor-implement lane-tracking system.

The two classes OglApp and GuiApp (Fig. 4) inherit from the openFrameworks Class ofBaseApp which generates a window and implements interaction methods for mouse, keyboard etc. Each one runs cyclically as an independent task for updating and rendering data. The following five Classes (Fig. 3) inherit from the openFrameworks Class ofThread, which allow them to run cyclically as an independent task as well. As already mentioned, these five classes represent the five
blocks found in the Block Diagram of Figure 1. The connection between these Classes takes place inside GuiApp using the event handling methods of openFrameworks.

Figure 3: Simplified UML diagram corresponding to the block diagram of Figure 1 used for the implement-tractor simulation based on openFrameworks.

2.2.1 Steering System Thread

The Class Gp_s_Thread implements the steering system of the tractor with the mathematical representation of Eq. (8). As shown in Figure 3, the set point, or curvatureIn, is set by the listener onNewCurInEvent and the output signal curvatureOut triggers the event newCurOutEvent. For managing matrix operations, a C++ template library for linear algebra called Eigen was used. The initialization of the matrices of the state space system takes place in the constructor and appears as follows:

\[
A = \begin{bmatrix}
1.8184, & 1, & 0, & 0, \\
-1.1828, & 0, & 1, & 0, \\
0.6562, & 0, & 0, & 1, \\
-0.3075, & 0, & 0, & 0,
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
0.000859 \\
-0.014128 \\
0.028873 \\
0.0
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
1, & 0, & 0, & 0
\end{bmatrix}
\]

\[
D = \begin{bmatrix}
0.0
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0
\end{bmatrix}
\]

The method threadedFunction is called cyclically with a sampling time of twenty milliseconds for this simulation and appears as follows:

```cpp
while(isThreadRunning()){
    dSet = asin(axisWidth/(1000/curvatureIn))*180/M_PI;
    u(0) = dSet;
    if(lock()){
        x.tail(n) = A*x.head(n) + B*u;
        y = C*x.head(n) + D*u;
        x.head(n) = x.tail(n);
        y = 1000/(axisWidth/sin(dOut*M_PI/180));
        ofNotifyEvent(newCurvatureOutEvent, curvatureOut, this);
        unlock();
    }else{
        ofLogWarning("SteeringSystemThread: ")
```
Where \( d\text{Set} \) and \( d\text{Out} \) are the steering angle in degrees since the identified model of Eq. (8) was performed using the input and output signals in degrees. For a sampling time of twenty milliseconds \( dT = 0.02 \).

### 2.2.2 Vehicle Dynamic Thread (Yaw Rate)

The Class \( G_{Ct}\_\text{Thread} \) implements the dynamics of the Implement-Tractor system represented in Equations (2) and (3). The functionality of this Class is very similar to the one of the steering system (section 2.2.1). The difference is that the digitalization of the state space takes place online. This allows the possibility of changing the data such as linear speed or cornering stiffness of the Implement on the fly. With this, one can observe how the dynamics of the system as well as the accuracy of the navigation system are affected by those changes. The matrix system is initialized with the values of Table 1 in the constructor as shown below:

```
a = 1.6965;
b = 3.9585;
c = 2.19;
Izz = 18500;
mass = 10830;
Caf = 2400;
Cah = 600;
Car = 5000;
dT = 0.02;
```

Then, the threadedFunction looks as follows:

```java
while(isThreadRunning()){
    if(Vx!=0){
        A(0,0) =-(Cah+Car+Caf)/(mass*Vx);
        A(0,1) =-Vx +
                 (Cah*(b+c)+Car*b-Caf*a)/(mass*Vx);
        A(1,0) =(-a*Caf+b*Car+(b+c)*Cah)/(Izz*Vx);
        A(1,1) =-1*(pow(a,2)*Car+pow(b,2)*Cah)/(Izz*Vx);
        B(0,0) = Caf/mass;
        B(1,0) = Caf*a/Izz;
        Az =  I + A*dT + A*A*pow(dT,2)/2;
        Bz = (I + A*dT/2)*dT*B;
        }
    }

    if(lock()){
        u(0) = asin(axisWidth/(1000/curvatureOut))
               *180/M_PI;
        y = C*x.head(n) + D*u + VectorXd::Random(1)*0.2;
        x.head(n) = Az*x.head(n) + Bz*u;
        rOut = y(0);
        ofNotifyEvent(
            newrOutEvent,rOut,this);
        unlock();
        ofSleepMillis(dT*1000);
    }else{
        ofLogWarning("VehicleDynamicThread: ")
        "threadedFunction() Unable to lock mutex."
    }
}
```

Where \( r\text{Out} \) is the yaw rate of the Implement-Tractor system and the input signal, which is taken from the Steering System (section 2.2.1) is again converted from curvature to steering angle in degrees.

The controller of the system consist of a feedback and a feedforward contained in the Class \( G_{Ct}\_\text{Thread} \) where the threadedFunction looks as follows:

```java
float error = rSet - rOut;
curvatureIn = Kff*rSet + Kpr*error;
ofNotifyEvent(
    newCurvatureInEvent,
curvatureIn,this);
```

Here, \( r\text{Set} \) is the desired yaw rate set point and it is updated through a listener connected to the controller of the lateral position (\( G_{Cl}\_\text{Thread} \)).

### 2.2.3 Lateral Position Thread

The Class \( G_{Pl}\_\text{Thread} \) has a listener of the yaw rate which is used to calculate the heading and the position in site coordinates using Eq. (9) to (11). The threadedFunction looks as follows:

```java
while(isThreadRunning()){
    if(lock()){
        y(0) = 0.0;
y(1) = 0.0;
rOut = y(0);
ofNotifyEvent(
            newrOutEvent,rOut,this);
    }else{
        y(0) = 0.0;
y(1) = 0.0;
rOut = y(0);
ofNotifyEvent(
            newrOutEvent,rOut,this);
    }
}
```

Here, \( r\text{Out} \) is the yaw rate of the Implement-Tractor system and the input signal, which is taken from the Steering System (section 2.2.1) is again converted from curvature to steering angle in degrees.

The controller of the system consist of a feedback and a feedforward contained in the Class \( G_{Ct}\_\text{Thread} \) where the threadedFunction looks as follows:

```java
float error = rSet - rOut;
curvatureIn = Kff*rSet + Kpr*error;
ofNotifyEvent(
    newCurvatureInEvent,
curvatureIn,this);
```

Here, \( r\text{Set} \) is the desired yaw rate set point and it is updated through a listener connected to the controller of the lateral position (\( G_{Cl}\_\text{Thread} \)).
if(lock()) {
  heading = rOut*dT + heading;
  courseAngle = heading - referenceAngle;
  Px = cos(heading*M_PI/180) * (Vx*dT) + Px;
  Py = sin(heading*M_PI/180) * (Vx*dT) + Py;
  lPos = sin(courseAngle*M_PI/180) * (Vx*dT) + yOut;
  ofNotifyEvent(
    newHeadingEvent, heading, this);
  ofNotifyEvent(
    newPxEvent, Px, this);
  ofNotifyEvent(
    newPyEvent, Py, this);
  ofNotifyEvent(
    newOutEvent, yOut, this);
  unlock();
} else{
  ofLogWarning(
    "controller_LateralPosition: ")
    "threadedFunction()
      Unable to lock mutex.";
}

ofSleepMillis(dT*1000);
}

2.2.4 Graphical User Interface

Figure 4 shows two Classes: GuiApp and OglApp. The first one generates a window (Fig. 5) which renders 2D graphics for the interaction with the system. It contains a control panel from the library ofDatGui with bar graphs for changing values such as the velocity of the tractor, the cornering stiffness or the controller parameters for the feedback, feed forward and PID (Fig. 7).

This function also calculates the lateral position lPos which corresponds to the position of the coordinate y1 of a local coordinate system rotated by a desired referenceAngle ψ (Fig. 9). Therefore, if the referenceAngle is zero, there is no rotation and the lateral position equals y in site coordinates. On the other hand if, for instance, the desired path lies 45 degrees with respect to x in site coordinates, the reference angle should be -45 degrees so the controller keeps the lateral position around zero i.e. the heading at 45 degrees at all times.

The controller of the system consists of a PID in the Class GCl_Thread in which the threaded function appears as follows:

while(isThreadRunning()){
  if(lock()){
    pError = lPosSet - lPosOut;
    iError += ek_1*dT;
    dError = (pError - ek_1)/dT;
    ek_1 = pError;
    rSet = yKlp*pError + yKli*iError + yKld*dError;
    ofNotifyEvent(
      newrSetEvent, rSet, this);
    unlock();
    ofSleepMillis(dT*1000);
  } else{
    ofLogWarning(
      "controller_LateralPosition: ")
      "threadedFunction()
        Unable to lock mutex.";
  }
}

Figure 5: Interactive Window generated by the Class GuiApp.

Figure 6: Plots of different values such as set point and measured curvature can be rendered from the Class GuiApp.

The way openFrameworks renders in 2D allows in a very easy way to plot the values of the desired signals as shown in Figure 6. GuiApp also contains an instance of the different parts of the navigation.
Figure 7: Interactive control panel using ofDatGui.

system (Figures 1 and 3) and the connection of their events and listeners takes place in its constructor as follows:

```cpp
// Vehicle steering and dynamics
ofAddListener(steeringSystem.newCurvatureOutEvent, &vehicleDynamic, &VehicleDynamicThread::onNewCurvatureOutEvent);

ofAddListener(steeringSystem.newCurvatureInEvent, &vehicleDynamic_ctrl, &SteeringSystemThread::onNewCurvatureInEvent);

// lateral position & controller
ofAddListener(vehiclePosition.newlPosEvent, &vehiclePosition_ctrl, &Controller_LateralPositionThread::onNewlPosEvent);

ofAddListener(vehiclePosition_ctrl.newrSetEvent, &vehicleDynamic_ctrl, &Controller_VehicleDynamicThread::onNewrSetEvent);
```

The second Class from Figure 4, OglApp, generates a window with a 3D ambient were the tractor is rendered using primitives such as boxes and cylinders (Fig. 8). The position of the tractor is updated with the data of the lateral position object which is a member of guiApp (Fig. 4).

For visualizing the scene, a virtual camera is used and it can be static, allowing the user to interact with mouse and keys for moving the 3D scene, or dynamic by following the vehicle at a defined distance which can be changed with the Control Panel (Fig. 7). openFrameworks allows to import textures, so one can use a model of a tractor to give a more real perspective of the vehicle instead of using boxes and cylinders.

Figure 8: OpenGL wrapped by openFrameworks make the generation of a 3D ambient easy and straightforward.

2.2.5 KML Way Points

Of course a navigation system consists of tracking a lane, or way line, with a set of way points. One way
of doing this is importing a KML file which contains a list of coordinate points in geodetic format: \(\text{latitude} (\lambda), \text{longitude} (\phi), \text{altitude} (h)\).

![Figure 9: Lateral position transformation to site coordinates for each segment of a way line.](image)

The first step will be to transform from geodetic to ECEF (Earth-centered Earth-fixed) coordinates with Eq. (13) which provides the fundamental parameters of WGS 84 revised in 1997 (Rovira Más, 2011) (Misra and Enge, 2006).

\[
\begin{align*}
    a &= 6378137.0 [m] \\
    b &= a \cdot (1 - f) = 6356752.3 [m] \\
    f &= \frac{a}{b} = 0.00335281 \\
    e &= \sqrt{a \cdot (2 - f)} = 0.0818 \\
    N_0(\lambda) &= \frac{\sqrt{1 - e^2 \sin^2 \lambda}}{\sin \lambda} \\
    X &= (N_0 + h) \cdot \cos(\lambda) \cdot \cos(\phi) \\
    Y &= (N_0 + h) \cdot \cos(\lambda) \cdot \sin(\phi) \\
    Z &= [N_0 \cdot (1 - e^2) + h] \cdot \sin(\lambda)
\end{align*}
\]

From ECEF coordinates \((X,Y,Z)\) we can transform to NED coordinates (North East Down) with the help of Eq. (14)

\[
\begin{bmatrix}
    X - X_0 \\
    Y - Y_0 \\
    Z - Z_0
\end{bmatrix} = \begin{bmatrix}
    \sin(\lambda) \cos(\phi) & -\sin(\lambda) \sin(\phi) & \cos(\lambda) \\
    -\sin(\phi) & \cos(\phi) & 0 \\
    -\cos(\lambda) \cos(\phi) & -\cos(\lambda) \sin(\phi) & -\sin(\lambda)
\end{bmatrix} \begin{bmatrix}
    N \\
    E \\
    D
\end{bmatrix}
\]

Where \(X_0, Y_0, Z_0\) correspond to the origin, or first geodetic point of the KML-File. The converted NED coordinates can be used for lane-tracking (Fig. 9) using a site coordinate system \(X_n\) translated to end of the \(n\) segment and rotated around \(\psi\) (reference angle of the \(n\) segment to the East axle). In this way \(y_n\) will be the lateral position to the desired path, and will be used as an input signal for the controller (Section 2.2.3). \(x_n\) indicates to update the segment \(n + 1\) when it changes from negative to positive.

3 RESULTS

The navigation system generated by the simulation represents very closely a real Tractor-Implement system. Also, the user can load a KML file with the navigation way-points or manually change the set points of lateral position and course angle. The user can as well easily interact with the system by changing parameters such as velocity, PID parameters, ground-vehicle interaction (Cornering Stiffness) and vehicle dimensions and visualize the impact those changes have on the navigation system.

4 CONCLUSION

This article presents a method for simulating an interactive Tractor-Implement navigation system using the C++ toolkit openFrameworks which contains, among others powerful libraries, OpenGL for rendering 2D and 3D vector graphics. Simulating an off-road navigation system with the use of openFrameworks not only facilitates the design and 3D representation of an off-road vehicle but also gives the user a better understanding of how the system will react in quasi-real conditions in different situations. It will also save time and resources in setting up the real system and tuning the controller compared to traditional hardware tuning in the field.

ACKNOWLEDGMENT

The authors would like to thank AGCO for the provision of a Fendt 939 tractor and the technical support of this research.
REFERENCES


