Experimental Software Framework for Hybrid Simulation

Andreas Schellenberg & Stephen Mahin

With contributions from:
Gregory Fenves, Yoshikazu Takahashi, Frank McKenna

Department of Civil and Environmental Engineering
University of California, Berkeley
Hybrid Simulation

\[ M \ddot{u} + C \dot{u} + P_r(u) = P(t) \]

Dynamic Loading
- Seismic
- Wind
- Blast/Impact
- Wave
- Traffic

analytical model of structural energy dissipation and inertia
physical model of structural resistance
Hybrid Simulation

- Model the well understood parts of a structure in a finite element program on one or more computers
- Leave the construction and testing of the highly nonlinear and/or numerically hard to model parts of the structure in one or more laboratories
- Can also be seen as an advanced form of component testing, where boundary conditions are correctly imposed
Advantages

- Enables dynamic testing of full-scale specimens
- Quasi-static testing equipment sufficient
- Fewer restrictions on size, weight and strength of a specimen
Advantages

- Geometric nonlinearities, three-dimensional effects, multi-support excitations and soil-structure interactions can be incorporated into the analytical model.

- Geographically distributed testing is made possible.
Main Challenge

- Lack of a common framework for development and deployment
- Problem specific implementations which are site and control system dependant
- Such highly customized software implementations are difficult to adapt to different structural problems

Need a robust, transparent, adaptable, and easily extensible framework for research and deployment
OpenFresco

- Open source Framework for Experimental Setup and Control
- Enable domain researchers to carry out Hybrid Simulations without specialized knowledge
- Allow IT and hybrid simulation specialists to extend frontiers of methodology, focusing only on their portions of interest
  - Facilitate additions and extensions for new equipment and procedures

Object-oriented programming approach
NEES-Compliant Deployment of OpenFresco

- No modification of numerical simulation framework is needed, other than the addition of new finite elements representing physical elements tested.

- Calls to obtain element stiffness, restoring force and other parameters made just like they would be in a numerical analysis, except they are executed physically somewhere on a local or wide area network.

- OpenFresco mediates in a modular and highly structured manner instructions between numerical simulation computer(s) and laboratory equipment.
OpenFresco Components

- **FE-Software**
  - Interfaces to the FE-Software, stores data and facilitates distributed testing.

- **Experimental Site**
  - Transforms between the experimental element degrees of freedom and the actuator degrees of freedom (linear vs. non-linear transformations).

- **Experimental Setup**

- **Experimental Control**
  - Interfaces to the different control and data acquisition systems in the laboratories.

- **Control System in Laboratory**
  - Local deployment.
OpenFresco Components

- RemoteExpSite
  - Exp.Setup
  - Exp.Control
  - Control System in Laboratory

- NTCPExpSite
  - Exp.Setup
  - Control Plugin with transformation
  - Control System in Laboratory

- NTCPExpSite
  - Exp.Setup
  - Control Plugin without transformation
  - Control System in Laboratory

- FE-Software
- network deployment

ActorExpSite
- Exp.Setup
- Control System in Laboratory

TCP/IP

NTCP
OpenFresco Data Transformation
Finite-Element Software

Currently using OpenSees; however, nearly any software allowing the addition of elements and having the appropriate communication channels can be used.

Furthermore, a Matlab client which is able to interface with OpenFresco is under development as well.
OpenSees Components

ModelBuilder constructs the objects in the model and adds them to the Domain.

Domain holds the state of the model at time $t_i$ and $t_i + dt$.

Recorder monitors user defined parameters in the model during the analysis.

Analysis moves the model from state at time $t_i$ to state at time $t_i + dt$. 

FE-Software

Experimental Site

Experimental Setup

Experimental Control

Control System in Laboratory
Direct Integration Methods

\[ M \cdot \ddot{u}_n + C \cdot \dot{u}_n + P_r (u_n) = P(t_n) \]

- Mass matrix \( M \) is often singular
  - second order differential equation infinitely stiff
  - fully implicit numerical methods

- Make as few function calls as possible
- Use constant Jacobian in the numerical methods since tangent stiffness is not available
Direct Integration Methods

Explicit Integrators
- explicit Newmark Method
- Central-Difference Method
- explicit Alpha-Method
- generalized explicit Alpha-Method

Implicit Integrators
- Newmark Method
- Alpha-Method
- generalized Alpha-Method
- Collocation Method
Direct Integration Methods

- Implicit Integrators with sub-stepping (constant number)
  - Newmark-HybridSimulation Method
  - generalized Alpha-HybridSimulation Method
  - Collocation-HybridSimulation Method

- Predictor-Corrector Integrators
  - Alpha-OS Method
  - generalized Alpha-OS Method

![Graph showing the relationship between \( P_i(u) \), \( u \), and \( u_{n-1}, u_n \).]
OpenSees Components

ModelBuilder constructs the objects in the model and adds them to the Domain.

Domain holds the state of the model at time $t_i$ and $t_i + dt$.

Recorder monitors user defined parameters in the model during the analysis.

Analysis moves the model from state at time $t_i$ to state at time $t_i + dt$.
OpenSees Components

- Domain
  - Node
  - Element
  - MP_Constraint
  - SP_Constraint

- ExperimentalElement
- BeamColumn

- OpenFRESCO
- Control System in Laboratory

FE-Software
- Experimental Site
- Experimental Setup
- Experimental Control

Control System in Laboratory
Experimental Elements

1) EETruss (2D,3D)

```
element expTruss $eleTag $iNode $jNode $siteTag
   -initStif $Kij <-iMod> <-rho $rho>
```

- `$eleTag` unique element tag
- `$iNode, $jNode` end nodes
- `$siteTag` tag of previously defined site object
- `$Kij` initial stiffness matrix element (1 x 1)
- `<-iMod>` flag for I-Modification (optional, default=false)
- `<-rho $rho>` mass per unit length (optional, default=0.0)
Experimental Elements

2) EEBeamColumn (2D,3D)

```
element expBeamColumn $eleTag $iNode $jNode $siteTag $tranTag -initStif $Kij <-iMod>
<-rho $rho>
```

- `$eleTag` unique element tag
- `$iNode,$jNode` end nodes
- `$siteTag` tag of previously defined site object
- `$tranTag` tag of previously defined crd-transf object
- `$Kij` initial stiffness matrix elements (ndf x ndf)
- `-iMod` flag for I-Modification (optional, default=false)
- `$rho` mass per unit length (optional, default=0.0)

Controlled displacements and acquired forces:
- $d_1, q_1$
- $d_2, q_2$
- $d_3, q_3$

Δx, Δ
Experimental Elements

3) EEZeroLength (2D,3D)

```
element expZeroLength $eleTag $iNode $jNode $siteTag
  -dir $dirs -initStif $Kij <-iMod>
  <-orient $x1 $x2 $x3 $y1 $y2 $y3>
```

- **$eleTag**: unique element tag
- **$iNode, $jNode**: end nodes
- **$siteTag**: tag of previously defined site object
- **$dirs**: force directions (1-3, 1-6)
- **$Kij**: initial stiffness matrix elements (nDir x nDir)
- **-iMod**: flag for I-Modification (optional, default=false)
- **$xi, $yi**: local x- and y-axis (optional, default=X,Y)

Controlled displacements and acquired forces:

```
\[ d_1, q_1 \]
\[ d_2, q_2 \]
\[ d_3, q_3 \]
```

FE-Software

Experimental Site
Experimental Setup
Experimental Control
Control System in Laboratory

FE-Software

nees@berkeley
Experimental Elements

4) EEChevronBrace (2D,3D)

```
element expChevronBrace $eleTag $iNode $jNode $kNode $siteTag -initStif $Kij <-iMod> <-rho1 $rho1> <-rho2 $rho2>
```

- `$eleTag` unique element tag
- `$iNode, $jNode` end nodes
- `$kNode` tag of previously defined site object
- `$siteTag` tag of previously defined site object
- `$Kij` initial stiffness matrix elements (ndf x ndf)
- `-iMod` flag for I-Modification (optional, default=false)
- `$rho1, $rho2` masses per unit length (optional, default=0.0)
Adding Experimental Elements

Public Member Functions

Constructor and Destructor

ExperimentalElement(int tag, int classTag, ExperimentalSite &theSite);
virtual ~ExperimentalElement();

Methods dealing with nodes and number of external dof

virtual int getNumExternalNodes(void) const = 0;
virtual const ID &getExternalNodes(void) = 0;
virtual Node **getNodePtrs(void) = 0;
virtual int getNumDOF(void) = 0;

Method to obtain basic dof size; equal to the max num dof that can be controlled

virtual int getNumBasicDOF(void) = 0;

Methods dealing with committed state and update

virtual int commitState(void);
virtual int update(void);
virtual bool isSubdomain(void);

Methods to set and to obtain the initial stiffness matrix

virtual int setInitialStiff(const Matrix& stiff) = 0;
const Matrix &getInitialStiff(void);

Methods to return the damping and mass matrices

virtual const Matrix &getDamp(void);
virtual const Matrix &getMass(void);

Methods for applying loads

virtual void zeroLoad(void) = 0;
virtual int addLoad(ElementalLoad *theLoad, double loadFactor) = 0;
virtual int addInertiaLoadToUnbalance(const Vector &accel) = 0;
setRayleighDampingFactors(double alphaM, double betaK, double betaK0, double betaKc);

Methods for obtaining resisting force (force includes elemental loads)

virtual const Vector &getResistingForce(void) = 0;
virtual const Vector &getResistingForceIncInertia(void);

Methods for obtaining information specific to an element

virtual Response *setResponse(const char **argv, int argc, Information &eleInformation);
virtual int getResponse(int responseID, Information &eleInformation);
Adding Experimental Elements

```cpp
int EESeanColumn2d::update( void )
{
    // get current time
    double time = this->getDomain()->getCurrentTime();
    // update the coordinate transformation
    theCoordTransf->update();
    // determine dep, vel and acc in basic system A
    const Vector &db = theCoordTransf->getBasicTrialDisp();
    const Vector &vb = theCoordTransf->getBasicTrialVel();
    const Vector &ab = theCoordTransf->getBasicTrialAccel();
    // transform displacements from basic sys A to basic sys B
    static Vector dbE(3), vbE(3), abE(3);
    dbE = T*db;
    vbE = T*vb;
    abE = T*ab;
    if ( dbE != 0 )
    {
        // compute the target displacement
        target = dbE;
        // set the trial responses at the site
        if ( isCopy == false )
            eSite->setTrialResponses(dbE, vbE, abE);
    }
    return 0;
}
```

```cpp
const Vector& EESeanColumn2d::getResistingForce( void )
{
    theCoordTransf->update();
    // zero the residual
    theVector.Zero();
    // get measured resisting forces
    static Vector qB(3);
    qB = eSite->getForce();
    // add initial forces
    for ( int i=0; i<3; i++ )
    {
        qB(i) += q[i];
    }
    if ( iMod == true )
    {
        // get measured displacements
        static Vector xmeasDisp(3);
        xmeasDisp = eSite->getDisp();
        // correct for displacement control errors using I-xodulation
        qB = kInit*(xmeasDisp - targDisp);
        // use elastic axial force if axial force from test is zero
        if ( qB(0) == 0 )
        {
            qB(0) = kInit(0,0) * targDisp(0);
        }
    }
    // transform from basic sys B to basic sys A
    q = T*qB;
    // Vector for reactions in basic system A
    Vector p0Vac(pB, 3);
    // determine resisting forces in global system
    theVector = theCoordTransf->getGlobalResistingForce( q, p0Vac );
    // subtract external load
    theVector.addVector(1.0, theLoad, -1.0);
    return theVector;
}
```

```cpp
int EESeanColumn2d::getInitialStiff( const Matrix &kInit )
{
    kInit = kInit;
    if ( kInit.noRows() != 3 || kInit.noCols() != 3 )
    {
        <ERROR> "EESeanColumn2d::getInitialStiff() - matrix size is incorrect for element: 
        " << this->getName() << endl;
        return -1;
    }
    // transform the stiffness from basic sys B to basic sys A
    static Matrix kInitA(3,3);
    kInitA = theCoordTransf->getInitialGlobalStiffMatrix( kInit );
    // transform the stiffness from basic to the global system
    theInitStiff.Zero();
    theInitStiff = theCoordTransf->getInitialGlobalStiffMatrix( kInitA );
    return 0;
}
```
Experimental Setups

1) ESNoTransformation

expSetup NoTransformation $tag $ctrlTag -dir $dirs
<-dspCtrlFact $dspCF> ...

$tag unique setup tag
$ctrlTag tag of previously defined control object
dirs directions (1-6)
Experimental Setups

2) ESOOneActuator

expSetup OneActuator $tag $ctrlTag $dir
<-dspCtrlFact $dspCF> ...

$tag unique setup tag
ctrlTag tag of previously defined control object
dir direction (1-6)
Experimental Setups

3) ESTwoActuators

\[
\text{expSetup } \text{TwoActuators } \$tag \$ctrlTag \$nlGeomFlag \\
\$La0 \$La1 \$L \leftarrow \text{dspCtrlFact } \$dspCF > \ldots
\]

$tag: unique setup tag
$ctrlTag: tag of previously defined control object
$nlGeomFlag: nonlinear geometry flag
$La0: length of actuator 0
$La1: length of actuator 1
$L: length of rigid link
Experimental Setups

4) ESThreeActuators

```plaintext
expSetup ThreeActuators $tag $ctrlTag $nlGeomFlag $La0 $La1 $La2 $L0 $L1
<-dspCtrlFact $dspCF> ...
```

- `$tag`: unique setup tag
- `$ctrlTag`: tag of previously defined control object
- `$nlGeomFlag`: nonlinear geometry flag
- `$La0`, `$La1`, `$La2`: length of actuators 0, 1, 2
- `$L0`, `$L1`: length of rigid links 0, 1
Experimental Setups

5) ES Chevron Brace Jnt Off

-expSetup Chevron Brace Jnt Off $tag $ctrlTag $nlGeomFlag $La0 $La1 $La2 $L0 $L1 $L2 $L3 $L4 <- dSpCtrlFact $dspCF> ...

-tag unique setup tag
-ctrlTag tag of previously defined control object
-nlGeomFlag nonlinear geometry flag
-La0 length of actuator 0
-La1 length of actuator 1
-La2 length of actuator 2
-L0 length of rigid link 0
-L1 length of rigid link 1
-L2 length of rigid link 2
-L3 length of rigid link 3
-L4 length of rigid link 4
Adding Experimental Setups

Public Member Functions

Constructor and Destructor

ExperimentalSetup(int tag, ExperimentalControl& theControl);
ExperimentalSetup(const ExperimentalSetup& es);
virtual ~ExperimentalSetup();

Methods dealing with data sizes
void setElmtDataSize(int s);
int getElmtDataSize();
void setCtrlDataSize(int s);
int getCtrlDataSize();
void setDaqDataSize(int s);
int getDaqDataSize();

Methods dealing with execution and acquisition
virtual int setup() = 0;
virtual int propose(const Vector& dsp, const Vector& vel, const Vector& acc) = 0;
virtual int execute() = 0;
virtual int commitState() = 0;
virtual int acquire() = 0;

Methods to obtain the response
const Vector& getDisp();
const Vector& getVel();
const Vector& getAccel();
const Vector& getForce();

Methods to set the control and data acquisition factors
void setDspCtrlFactor(const Vector& f);
void setVelCtrlFactor(const Vector& f);
void setAccCtrlFactor(const Vector& f);
void setDspDaqFactor(const Vector& f);
void setFrcDaqFactor(const Vector& f);

Method to get a copy
virtual ExperimentalSetup *getCopy (void) = 0;

int propose( )

int acquire( )
Adding Experimental Setups

```c
int ESTwoActuators2d::propose(const Vector4 dsp, const Vector4 vel,
const Vector4 acc)
{
    // linear geometry
    if (nFlag == 0) {
        // actuator 0
        (*dspCtrl)(0) = (*dspCtrlFact)(0) * (-dsp(1));
        (*velCtrl)(0) = (*velCtrlFact)(0) * (-vel(1));
        (*accCtrl)(0) = (*accCtrlFact)(0) * (-acc(1));
        // actuator 1
        (*dspCtrl)(1) = (*dspCtrlFact)(1) * (-dsp(1) - L * dsp(2));
        (*velCtrl)(1) = (*velCtrlFact)(1) * (-vel(1) - L * vel(2));
        (*accCtrl)(1) = (*accCtrlFact)(1) * (-acc(1) - L * acc(2));
    }
    // nonlinear geometry
    else if (nFlag == 1) {
        operr << "ESTwoActuators2d::propose() - nonlinear geometry not implemented yet";
        return ExperimentalReturnTypeFailed;
    }
    return ExperimentalReturnType_ready;
}
```

```c
int ESTwoActuators2d::acquire()
{
    theControl->acquire(dspDaq, frcDaq);
    // linear geometry
    if (nFlag == 0) {
        // measured displacements
        (*dspInt)(0) = 0;
        (*dspInt)(1) = -(*dspFact)(0) * (*dspDaq)(0); // measured forces
        (*frcInt)(0) = 0;
        (*frcInt)(1) = -(*frcFact)(0) * (*frcDaq)(0);
    }
    // nonlinear geometry
    else if (nFlag == 1) {
        operr << "ESTwoActuators2d::acquire() - nonlinear geometry not implemented yet";
        return ExperimentalReturnTypeFailed;
    }
    return ExperimentalReturnType_completed;
}
```
Experimental Controls

1) ECdSpace

```
expControl dSpace $tag $numSetups "type"
   "boardName"
```

- `$tag` unique control tag
- `$numSetups` number of setups
- "type" predictor-corrector type
- "boardName" name of dSpace board
Experimental Controls

2) ECxPCTarget

```
expControl xPCtarget $tag $numSetups $type "ipAddr"
  "ipPort" "appName" "appPath"
```

$tag unique control tag
$numSetups number of setups
$type predictor-corrector type
"ipAddr" IP address of xPC Target
"ipPort" IP port of xPC Target
"appName" name of Simulink application to be loaded
"appPath" path to Simulink application
Experimental Controls

3) ECSrcramNet

expControl ScramNet $tag $numSetups ...

$tag unique control tag
$numSetups number of setups

... under development
Experimental Controls

4) ECNIER series

expControl NIEseries $tag $numCtrl $device

$tag unique control tag
$numSetups number of setups
$device id of device
Adding Experimental Controls

Public Member Functions

Constructor and Destructor

ExperimentalControl(int tag, int nCtrl, int nDaq);
ExperimentalControl(const ExperimentalControl& ec);
virtual ~ExperimentalControl();

Methods dealing with data sizes

int getCtrlDataSize();
int getDaqDataSize();

Methods to set and obtain the responses

virtual int setup() = 0;
virtual int execute(const Vector& dsp, const Vector& vel, const Vector& acc) = 0;
virtual int commitState();
virtual int acquire(Vector *dspDaq, Vector *frcDaq) = 0;

Method to add a data filter

void addFilter(SignalFilter & f);

Method to get a copy

virtual ExperimentalControl *getCopy (void) = 0;
Adding Experimental Controls

```c
int getTargDep(vector<ref Vector&> dep, const Vector& vel, const Vector& acc) {
    if (fabs(vel[0]) > 1e-3 || fabs(vel[1]) > 1e-3) {
        return 1;
    }
    else {
        return 0;
    }
}
```

```c
int getCtrlDep(vector<ref Vector&> dep, const Vector& vel, const Vector& acc) {
    if (fabs(vel[0]) > 1e-3 || fabs(vel[1]) > 1e-3) {
        return 1;
    }
    else {
        return 0;
    }
}
```

```c
int getFilter(vector<ref Vector&> dep, const Vector& vel, const Vector& acc) {
    if (fabs(vel[0]) > 1e-3 || fabs(vel[1]) > 1e-3) {
        return 1;
    }
    else {
        return 0;
    }
}
```
Conclusions

- Environment-independent framework for development and deployment will boost the use of hybrid simulation (on-site and tele-operation)

- Modularity and transparency of the framework permits existing components to be modified and new components to be added without much dependence on other objects.
  - Speed development of refined hybrid simulation procedures
Conclusions

- Large library of hybrid simulation direct integration methods, experimental elements, controller models, and event-driven solution strategies will be available to the user to choose from or adapt.

- Need:
  - User-community input of parameter passage and features
  - User feedback
  - NEESit assistance in streamlining network communications
Thank you!

Development and operation of the nees@berkeley Equipment Site is sponsored by NSF George E. Brown Jr. NEES grants.

http://nees.berkeley.edu