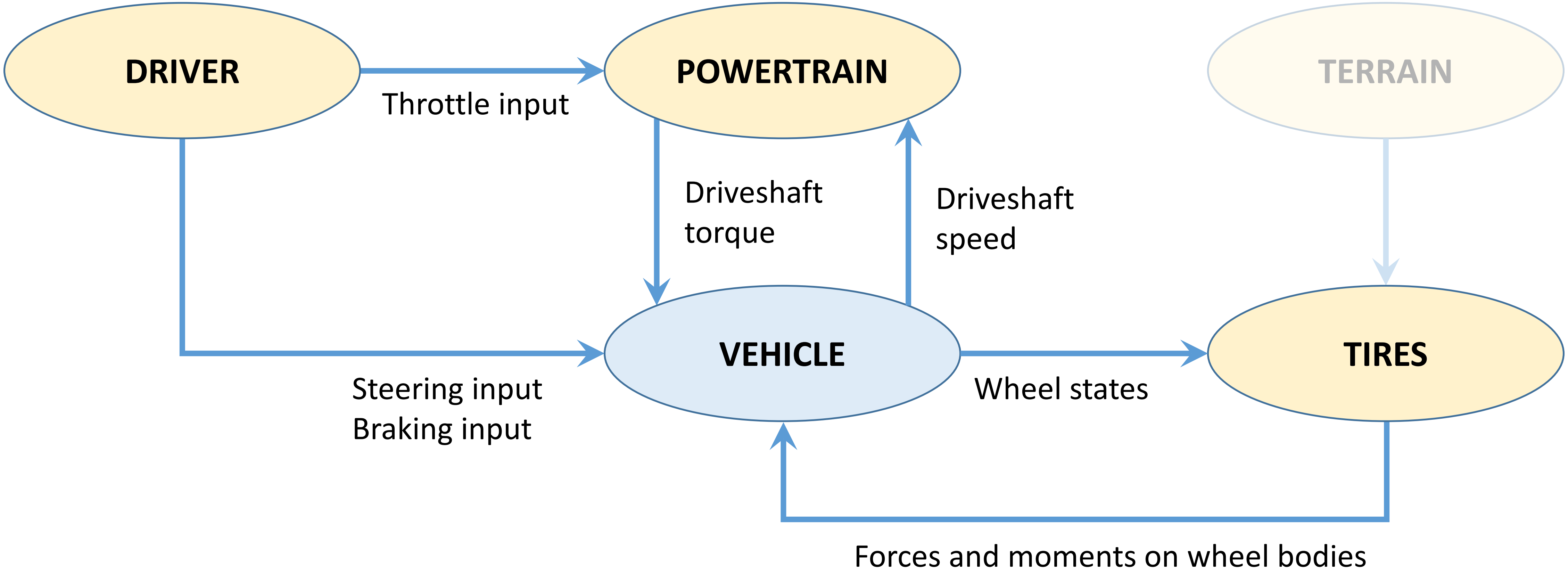
Chrono::Vehicle Tutorial

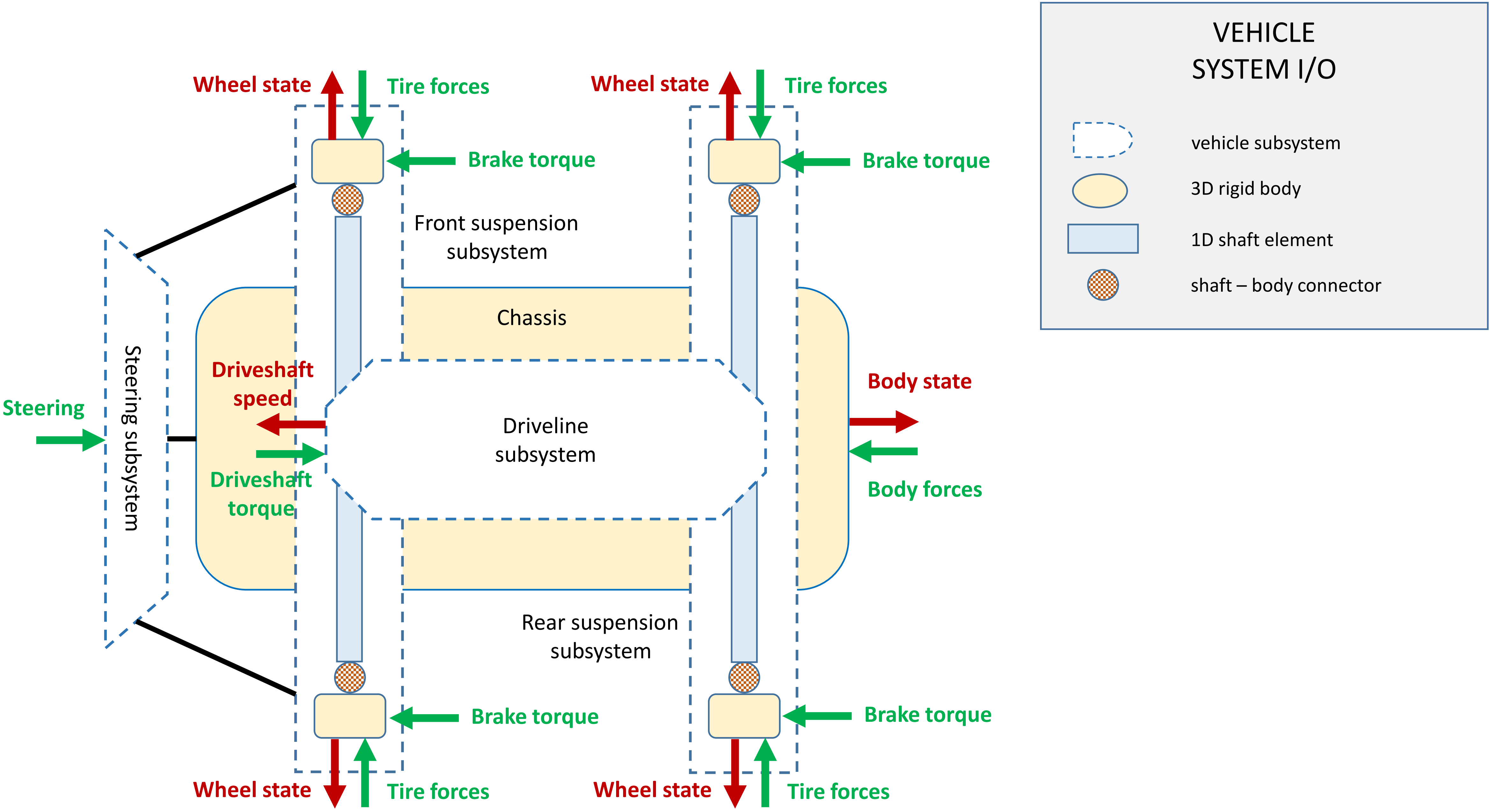
Wheeled vehicle system

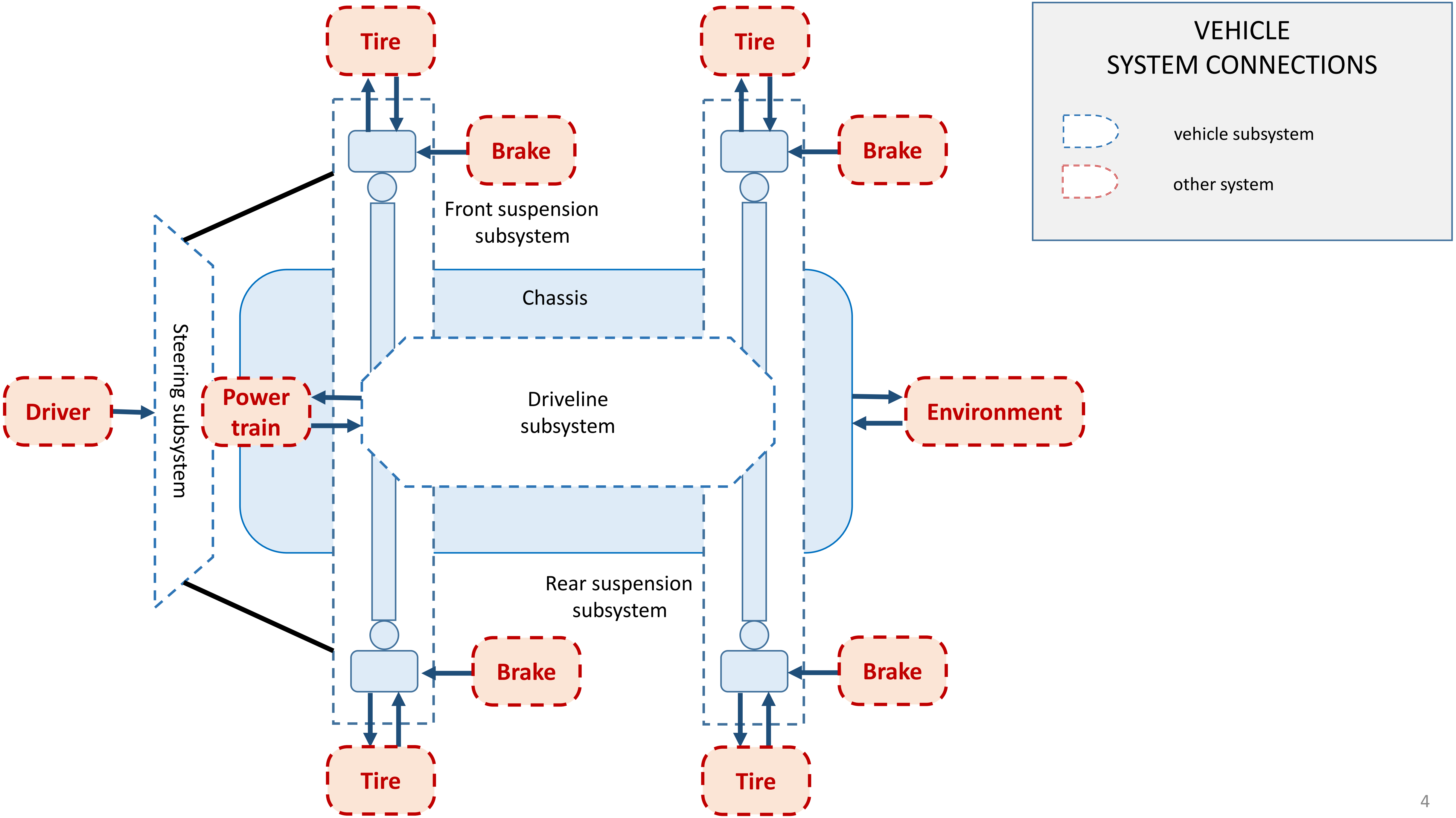
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

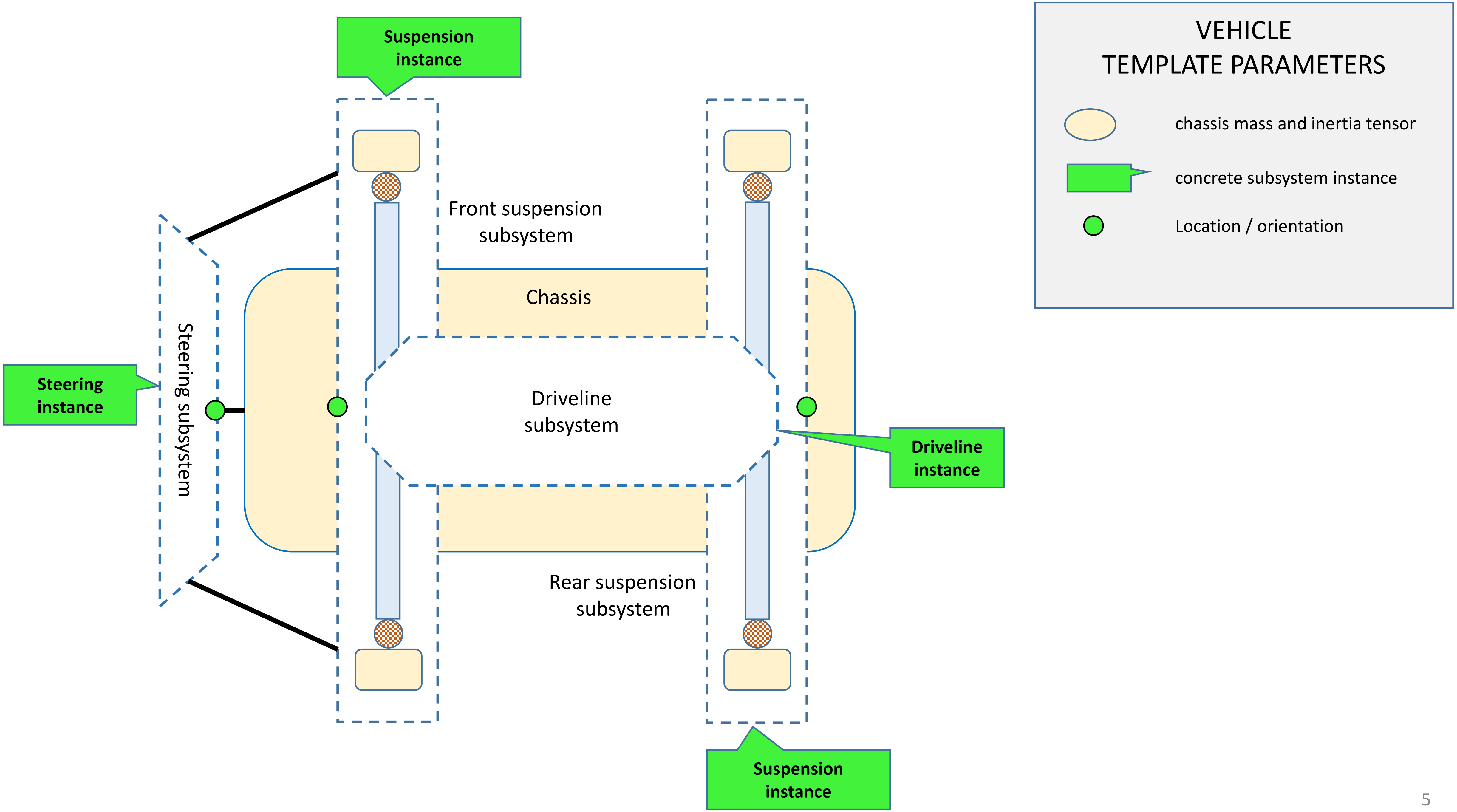
UNIVERSITÀ DEGLI STUDI DI PARMA

Data flow

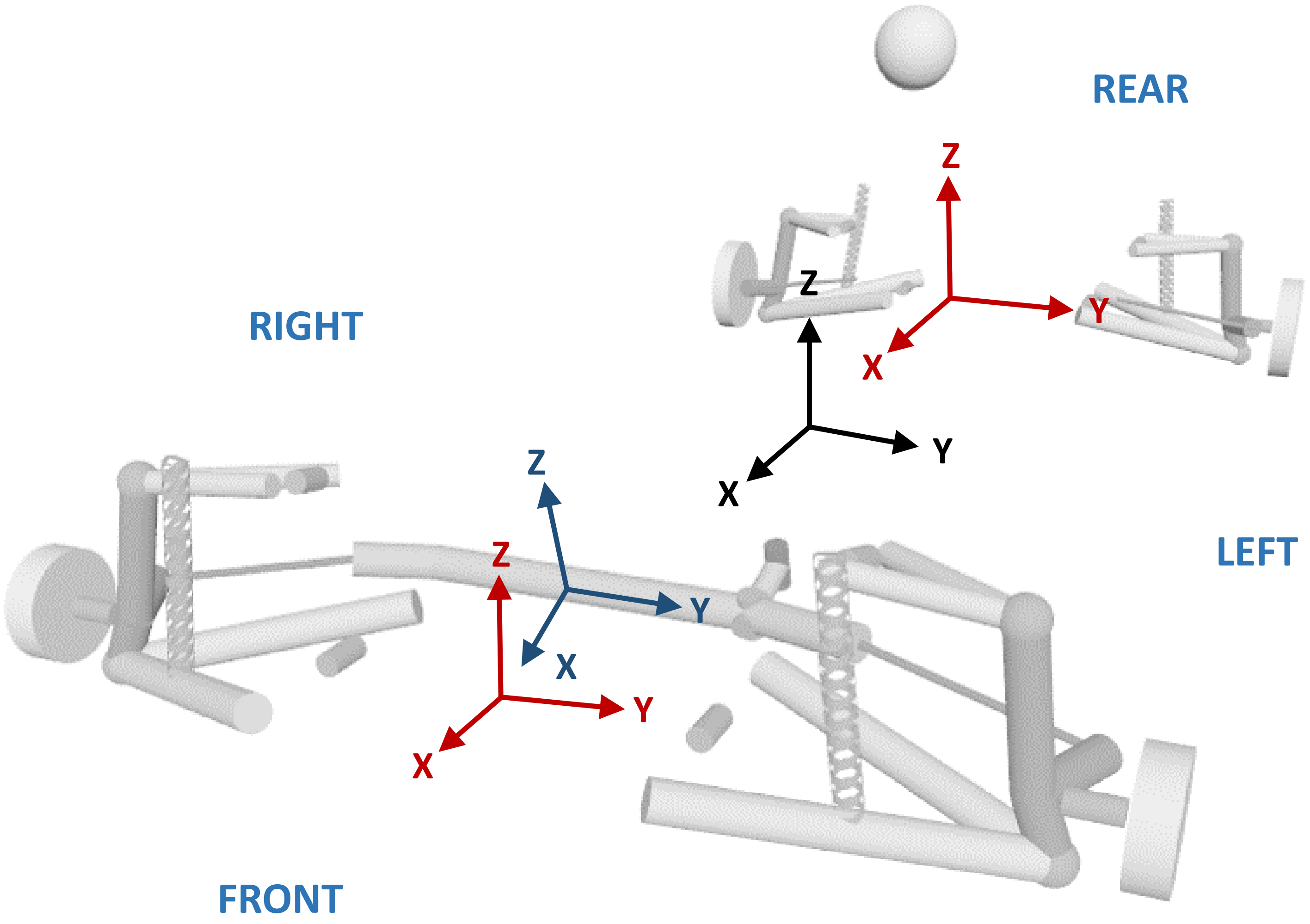

DRIVER
Throttle input
POWERTRAIN
TERRAIN
Driveshaft torque
Driveshaft speed
VEHICLE
TIRES
Steering input
Braking input
Wheel states
Forces and moments on wheel bodies


VEHICLE
SYSTEM I/O
vehicle subsystem
3D rigid body
1D shaft element
shaft – body connector
Wheel state
Tire forces
Wheel state
Tire forces
Brake torque
Brake torque
Front suspension
subsystem
Chassis
Steering subsystem
Steering
Driveshaft
speed
Driveshaft
torque
Driveline
subsystem
Body state
Body forces
Rear suspension
subsystem
Brake torque
Brake torque
Wheel state
Tire forces
Wheel state
Tire forces


VEHICLE
SYSTEM CONNECTIONS
vehicle subsystem
other system
Tire
Tire
Brake
Brake
Front suspension
subsystem
Chassis
Steering subsystem
Driver
Power
train
Driveline
subsystem
Environment
Rear suspension
subsystem
Brake
Brake
Tire
Tire


VEHICLE
TEMPLATE PARAMETERS
chassis mass and inertia tensor
concrete subsystem instance
Location / orientation
Suspension instance
Front suspension subsystem
Chassis
Steering instance
Steering subsystem
Driveline subsystem
Driveline instance
Rear suspension subsystem
Suspension instance

Vehicle ISO reference frames



(XYZ) – chassis reference frame
(XYZ) – suspension reference frame
(XYZ) – steering reference frame

ChWheeledVehicle base class

- A ChWheeledVehicle is a Chrono ChVehicle:

```
/// Base class for chrono wheeled vehicle systems.
/// This class provides the interface between the vehicle system and other
/// systems (tires, driver, etc.).
/// The reference frame for a vehicle follows the ISO standard: Z-axis up, X-axis
/// pointing forward, and Y-axis towards the left of the vehicle.
class CH_VEHICLE_API ChWheeledVehicle : public ChVehicle
```

- A ChWheeledVehicle has:

```
ChSuspensionList m_suspensions;           ///< list of handles to suspension subsystems
ChAntirollbarList m_antirollbars;         ///< list of handles to antirollbar subsystems (optional)
std::shared_ptr<ChDriveline> m_driveline; ///< handle to the driveline subsystem
ChSteeringList m_steerings;               ///< list of handles to steering subsystems
ChWheelList m_wheels;                     ///< list of handles to wheel subsystems
ChBrakeList m_brakes;                     ///< list of handles to brake subsystems
```

ChWheeledVehicle base class accessors

- Deferring to its constituent subsystems as needed, a ChWheeledVehicle provides accessors for:
  - Vehicle subsystems
  - States of the vehicle’s wheel bodies (the suspension spindle bodies)
  - Inherits all accessors from ChVehicle

- A ChWheeledVehicle intermediates communication between other systems (e.g., tires, driver, etc.) and constituent subsystems (e.g., suspensions, brakes, etc.)

ChWheeledVehicle base class virtual functions

- Synchronize the vehicle at a communication time with data from other systems

```
/// Synchronize the state of this vehicle at the current time.
/// The vehicle system is provided the current driver inputs (throttle between
/// 0 and 1, steering between -1 and +1, braking between 0 and 1), the torque
/// from the powertrain, and tire forces (expressed in the global reference
/// frame).
virtual void Synchronize(
  double              time,               ///< [in] current time
  double              steering,           ///< [in] current steering input [-1,+1]
  double              braking,            ///< [in] current braking input [0,1]
  double              powertrain_torque,  ///< [in] input torque from powertrain
  const ChTireForces& tire_forces         ///< [in] vector of tire force structures
  ) {}
```

Data exchange structures

- WheelID class – encodes the ID of a vehicle wheel
- By convention, wheels are counted front to rear and left to right
    - For a vehicle with 2 axles, the order is: front-left, front-right, rear-left, rear-right
- A wheel ID encodes an axle number (0,1,2,…) and the vehicle side (0: left, 1: right)

$$id = 2 \, * axle + side$$

```
WheelID(int id) : m_id(id), m_axle(id / 2), m_side(VehicleSide(id % 2)) {}
WheelID(int axle, VehicleSide side) : m_id(2 * axle + side), m_axle(axle), m_side(side) {}

/// Return the wheel ID.
int id() const { return m_id; }

/// Return the axle index for this wheel ID.
/// Axles are counted from the front of the vehicle.
int axle() const { return m_axle; }

/// Return the side for this wheel ID.
/// By convention, left is 0 and right is 1.
VehicleSide side() const { return m_side; }
```

Data exchange structures

- BodyState structure – encapsulates the full state of a Chrono body
  - Position – as a 3D vector
  - Orientation – as a unitary quaternion
  - Linear velocity (with respect to the global frame) – as a 3D vector
  - Angular velocity (with respect to the global frame) – as a 3D vector

```
///
/// Structure to communicate a full body state.
///
struct BodyState {
  ChVector<>     pos;      ///< global position
  ChQuaternion<> rot;      ///< orientation with respect to global frame
  ChVector<>     lin_vel;  ///< linear velocity, expressed in the global frame
  ChVector<>     ang_vel;  ///< angular velocity, expressed in the global frame
};
```

Data exchange structures

- WheelState structure – encapsulates the full state of a wheel body
  - Position, orientation, velocity – same as BodyState
  - Additionally contains the wheel angular speed about its rotation axis – as a double scalar
- A wheel state structure is passed to each tire system during the inter-system communication phase

```
///
/// Structure to communicate a full wheel body state.
/// In addition to the quantities communicated for a generic body, the wheel
/// state also includes the wheel angular speed about its axis of rotation.
///
struct WheelState {
  ChVector<>     pos;      ///< global position
  ChQuaternion<> rot;      ///< orientation with respect to global frame
  ChVector<>     lin_vel;  ///< linear velocity, expressed in the global frame
  ChVector<>     ang_vel;  ///< angular velocity, expressed in the global frame
  double         omega;    ///< wheel angular speed about its rotation axis
};
```

Data exchange structures

- TireForce structure – encapsulates the tire forces applied to a wheel body
  - Force vector and application point (expressed in the global reference frame)
  - Moment vector (expressed in the global reference frame)
- A tire force structure is obtained from each tire system during the inter-system communication phase
  - The tire force and moment are applied to the wheel (spindle) body as external forces

```
///
/// Structure to communicate a set of generalized tire forces.
///
struct TireForce {
  ChVector<> force;       ///< force vector, expressed in the global frame
  ChVector<> point;       ///< global location of the force application point
  ChVector<> moment;      ///< moment vector, expressed in the global frame
};
```

JSON specification file for a wheeled vehicle (1/3)

```
{
  "Name":     "Test vehicle - 3 axles",
  "Type":     "Vehicle",
  "Template": "WheeledVehicle",

  "Chassis":
  {
    "Input File":     "generic/chassis/Chassis.json"
  },
```

System type (string)

Template type (string)

JSON file with chassis specification
(relative to the root of the data directory)

JSON specification file for a vehicle (2/3)

```
"Axles":
 [
   {
     "Suspension Input File":   "hmmwv/suspension/HMMWV_DoubleWishboneFront.json",
     "Suspension Location":     [1.6914, 0, 0.0264],
     "Left Wheel Input File":   "hmmwv/wheel/HMMWV_Wheel_FrontLeft.json",
     "Right Wheel Input File":  "hmmwv/wheel/HMMWV_Wheel_FrontRight.json",
     "Left Brake Input File":   "hmmwv/brake/HMMWV_BrakeSimple_Front.json",
     "Right Brake Input File":  "hmmwv/brake/HMMWV_BrakeSimple_Front.json"
   },

   {
     "Suspension Input File":   "generic/suspension/SolidAxleRear.json",
     "Suspension Location":     [-0.5, 0, 0.0264],
     "Left Wheel Input File":   "hmmwv/wheel/HMMWV_Wheel_RearLeft.json",
     "Right Wheel Input File":  "hmmwv/wheel/HMMWV_Wheel_RearRight.json",
     "Left Brake Input File":   "hmmwv/brake/HMMWV_BrakeSimple_Rear.json",
     "Right Brake Input File":  "hmmwv/brake/HMMWV_BrakeSimple_Rear.json"
   },

   {
     "Suspension Input File":   "generic/suspension/SolidAxleRear.json",
     "Suspension Location":     [-1.8, 0, 0.0264],
     "Left Wheel Input File":   "hmmwv/wheel/HMMWV_Wheel_RearLeft.json",
     "Right Wheel Input File":  "hmmwv/wheel/HMMWV_Wheel_RearRight.json",
     "Left Brake Input File":   "hmmwv/brake/HMMWV_BrakeSimple_Rear.json",
     "Right Brake Input File":  "hmmwv/brake/HMMWV_BrakeSimple_Rear.json"
   }

 ],
```

Offset of the suspension reference frame with respect to the chassis reference frame

JSON specification file for a vehicle (3/3)

```
 "Steering":
  {
    "Input File":            "hmmwv/steering/HMMWV_PitmanArm.json",
    "Location":              [1.24498, 0, 0.109322],
    "Orientation":           [0.98699637, 0, 0.16074256, 0],
    "Suspension Index":      0
  },

  "Driveline":
  {
    "Input File":            "hmmwv/driveline/HMMWV_Driveline4WD.json",
    "Suspension Indexes":    [0, 2]
  }
}
```

Position and orientation of the steering subsystem reference frame with respect to the chassis reference frame.

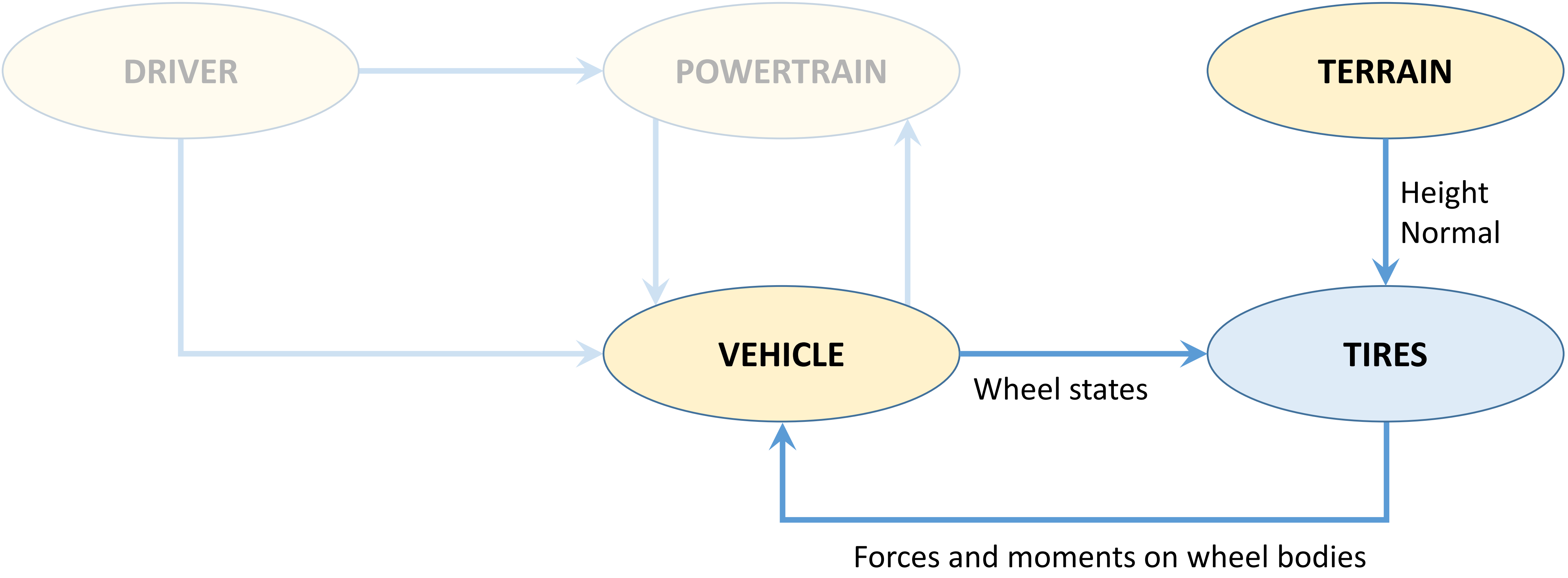
Index of the vehicle axle that is connected to the steering subsystem (here, the first axle in the list).

Indexes of the vehicle axles that are connected to the driveline subsystem (here, the first and third in the list). Must be consistent with the driveline type.

Tire Models

Rigid Tire
LuGre Tire
Pacejka Tire
ANCF Tire

Data flow


DRIVER
POWERTRAIN
TERRAIN
Height
Normal
VEHICLE
Wheel states
TIRES
Forces and moments on wheel bodies

ChTire base class

- Defines the common interface for any tire system
- All classes defining particular tire templates inherit from ChTire

```
///
/// Base class for a tire system.
/// A tire subsystem is a force element. It is passed position and velocity
/// information of the wheel body and it produces ground reaction forces and
/// moments to be applied to the wheel body.
///
class CH_VEHICLE_API ChTire : public ChPart
```

ChTire base class members

- A ChTire has:

```
VehicleSide m_side;                ///< tire mounted on left/right side
std::shared_ptr<ChBody> m_wheel;   ///< associated wheel body
```

ChTire base class virtual methods

- Update the tire at a communication time with data from other systems

```
/// Update the state of this tire system at the current time.
/// The tire system is provided the current state of its associated wheel and
/// a handle to the terrain system.
virtual void Synchronize(double time,                         ///< [in] current time
                         const WheelState& wheel_state,  ///< [in] current state of associated wheel body
                         const ChTerrain& terrain        ///< [in] reference to the terrain system
                         ) {}
```

- Advance the state of the tire to the next communication point

```
/// Advance the state of this tire by the specified time step.
virtual void Advance(double step) {}
```

- Set the (output) torque to be applied to the associated wheel body

```
/// Get the tire force and moment.
/// This represents the output from this tire system that is passed to the
/// vehicle system.  Typically, the vehicle subsystem will pass the tire force
/// to the appropriate suspension subsystem which applies it as an external
/// force one the wheel body.
virtual ChTireForce GetTireForce() const = 0;
```

Tire Models

Rigid Tire

ChRigidTire and RigidTire

- ChRigidTire is an abstract class (i.e., system template)
  - Define basic functions for pure virtual functions (only GetTireForce)
  - Define virtual functions that are common for all RigidTire models
    - Initialize(ChSharedPtr<ChBody> wheel)
    - virtual and pure virtual functions

- RigidTire is a concrete class
  - Cylinder collision geometry, constant tire inertia
  - Interacts with terrain using rigid body frictional contact
  - Implements pure virtual functions in ChRigidTire
  - Defines how to read input JSON file

```
class ChRigidTire: public ChTire;

class RigidTire: public ChRigidTire;
```

JSON specification file for rigid tire

```
{
  "Name":                         "HMMWV Rigid Tire",
  "Type":                         "Tire",
  "Template":                     "RigidTire",

  "Radius":                       0.4699,
  "Width":                        0.254,
  "Coefficient of Friction":      0.7
}
```

System type (string)

Template type (string)

Tire Models

LuGre Tire

ChLugreTire and LugreTire

- ChLugreTire is an abstract class (i.e., system template)
  - virtual functions that are common for all Lugre tire models:

```
virtual int getNumDiscs() = 0;
virtual double getRadius() = 0;
virtual const double* getDiscLocations() = 0;
virtual double getNormalStiffness() = 0;
virtual double getNormalDamping() = 0;
virtual void SetLugreParams() = 0;
```

- LugreTire is a concrete class
  - Defines how to read tire parameters from input JSON file
  - Tangent/Traction forces according to first order ODE
  - Documentation – SBEL Tech Report 2014-15 “Lugre Tire Model for HMMWV”, Aki Mikkola
  - sbel.wisc.edu/Publications

JSON specification file for LuGre tire

```
{
  "Name":             "HMMWV Lugre Tire",
  "Type":             "Tire",
  "Template":         "LugreTire",

  "Radius":            0.4699,

  "Disc Locations":    [-0.127, 0, 0.127],

  "Normal Stiffness": 2e6,
  "Normal Damping":    1e3,

  "Lugre Parameters" :
  {
    "sigma0":          [181.0,  60.0],
    "sigma1":          [  1.0,   0.2],
    "sigma2":          [  0.02,  0.002],
    "Fc":              [  0.6,   0.6],
    "Fs":              [  1.0,   1.0],
    "vs":              [  3.5,   3.5]
  }
}
```

System type (string)

Template type (string)

Transversal locations of the constituent discs, relative to the tire's center plane. This particular tire uses 3 equally spaced discs.

Stiffness and (viscous) damping used in calculating the normal tire force

LuGre tire parameters. Two values are specified for each parameter, the first for the longitudinal direction, the second for the transversal direction.

Tire Models

Pacejka Tire

ChPacejkaTire

- ChPacejkaTire is a **concrete class**
  - Could benefit from the abstract/concrete organization of previous two models
  - Allow older Pacejka formulations to be tested
  - Pac2002 formulation is ubiquitous in commercial software

- Chrono::Vehicle implementation tested and validated against commercial solutions
- Documentation: SBEL Tech Reports 2014-14 and 2014-16
  - "Validation of a Steady-State Magic Formula Tire in Chrono with a Comercial Software Implementation", J. Madsen and A. Dirr.
  - "Validation of a Single Contact Point Tire Model Based on the Transient Pacejka Model in the Open-Source Dynamics Software Chrono", J. Madsen
  - sbel.wisc.edu/Publications

Tire Models

Deformable tires

ANCF Tire

ChDeformableTire base class

- Base class for tires modeled with finite element meshes
- Derived classes: ChANCFTire, CHFEATire
- Provides support for defining:
  - FEA mesh
  - Connection to rim body
  - Internal pressure load
  - Contact geometry (contact surface) and contact material properties

ChDeformableTire base class members

```
std::shared_ptr<fea::ChMesh> m_mesh;                               ///< tire mesh
std::shared_ptr<ChLoadContainer> m_load_container;                 ///< load container (for pressure load)
std::vector<std::shared_ptr<fea::ChLinkPointFrame>> m_connections;  ///< tire-wheel point connections
std::vector<std::shared_ptr<fea::ChLinkDirFrame>> m_connectionsD;   ///< tire-wheel direction connections
std::vector<std::shared_ptr<ChLinkMateFix>> m_connectionsF;         ///< tire-wheel fix connection (point+rotation)

bool m_connection_enabled;  ///< enable tire connections to rim
bool m_pressure_enabled;    ///< enable internal tire pressure
bool m_contact_enabled;     ///< enable tire-terrain contact

double m_pressure;  ///< internal tire pressure

ContactSurfaceType m_contact_type;  ///< type of contact surface model (node cloud or mesh)
double m_contact_node_radius;       ///< node radius (for node cloud contact surface)
double m_contact_face_thickness;    ///< face thickness (for mesh contact surface)

float m_friction;       ///< contact coefficient of friction
float m_restitution;    ///< contact coefficient of restitution
float m_young_modulus;  ///< contact material Young modulus
float m_poisson_ratio;  ///< contact material Poisson ratio
float m_kn;             ///< normal contact stiffness
float m_gn;             ///< normal contact damping
float m_kt;             ///< tangential contact stiffness
float m_gt;             ///< tangential contact damping

std::shared_ptr<ChMaterialSurfaceDEM> m_contact_mat;           ///< tire contact material
std::shared_ptr<fea::ChVisualizationFEAmesh> m_visualization;  ///< tire mesh visualization
```

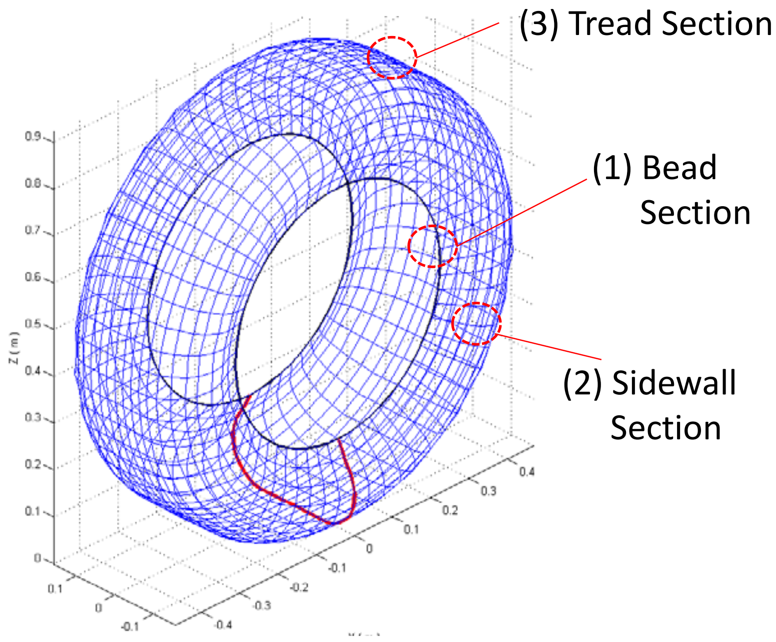
ChDeformableTire base class virtual methods

```
/// Create the FEA nodes and elements.
/// The wheel rotational axis is assumed to be the Y axis.
virtual void CreateMesh(const ChFrameMoving<>& wheel_frame,  ///< [in] frame of associated wheel
                        VehicleSide side                     ///< [in] left/right vehicle side
                        ) = 0;

/// Create the ChLoad for applying pressure to the tire.
/// A derived class must create a load and add it to the underlying load container.
virtual void CreatePressureLoad() = 0;

/// Create the contact surface for the tire mesh.
/// A derived class must create a contact surface and add it to the underlying mesh.
virtual void CreateContactSurface() = 0;

/// Create the tire-rim connections.
/// A derived class must create the various constraints between the tire and the
/// provided wheel body and add them to the underlying system.
virtual void CreateRimConnections(std::shared_ptr<ChBody> wheel  ///< [in] associated wheel body
                                  ) = 0;
```

ChANCFTire

User-specified parameters:
- Tire shape / size
- Layer properties
- Element mesh resolution

(1) Bead section	Cord angle	Thickness
Carcass	$\alpha = 90°$	$h = 0.5mm$
Rubber	$\alpha = 0°$	$h = 5.0mm$
Carcass	$\alpha = 90°$	$h = 0.5mm$
(2) Sidewall section		
Carcass	$\alpha = 90°$	$h = 0.5mm$
Rubber	$\alpha = 0°$	$h = 0.1mm$
Carcass	$\alpha = 90°$	$h = 0.5mm$
(3) Tread section		
Rubber	$\alpha = 0°$	$h = 1.0mm$
Belt	$\alpha = -20°$	$h = 0.3mm$
Belt	$\alpha = 20°$	$h = 0.3mm$
Carcass	$\alpha = 90°$	$h = 0.5mm$



- ANCF Tire model is connected to a rigid rim body through displacement and gradient constraints for the nodes at the outer edges of the bead section of the tire model.
- The internal air pressure is applied through a distributed load class available in Chrono::FEA.

JSON specification file for ANCF tire (1/2)


```
{
  "Name":            "HMMWV ANCF Tire",
  "Type":            "Tire",
  "Template":        "ANCFTire",

  "Tire Radius":     0.4673,
  "Rim Radius":      0.2683,
  "Rim Width":       0.254,

  "Contact Material":
  {
      "Coefficient of Friction":    0.9,
      "Coefficient of Restitution": 0.1,

      "Properties":
      {
          "Young Modulus":              2e6,
          "Poisson Ratio":              0.3
      },

      "Coefficients":
      {
          "Normal Stiffness":           2.0e6,
          "Normal Damping":             1.3e1,
          "Tangential Stiffness":       1.0e6,
          "Tangential Damping":         0
      }
  },
```

```
"Materials":
 [
     {
        "Type":     "Orthotropic",
        "Density": 0.1000000E+04,
        "E":        [0.7560000E+10 , 0.4740000E+08 , 0.4740000E+08],
        "nu":       [0.4500000E+00 , 0.4500000E+00 , 0.4500000E+00],
        "G":        [0.1634483E+08 , 0.1634483E+08 , 0.1634483E+08]
     },

     {
        "Type":     "Orthotropic",
        "Density": 0.2639000E+04,
        "E":        [0.1800000E+12 , 0.4740000E+08 , 0.4740000E+08],
        "nu":       [0.4500000E+00 , 0.4500000E+00 , 0.4500000E+00],
        "G":        [0.1634483E+08 , 0.1634483E+08 , 0.1634483E+08]
     },

     {
        "Type":     "Orthotropic",
        "Density": 0.1100000E+04,
        "E":        [0.4740000E+08 , 0.4740000E+08 , 0.4740000E+08],
        "nu":       [0.4500000E+00 , 0.4500000E+00 , 0.4500000E+00],
        "G":        [0.1634483E+08 , 0.1634483E+08 , 0.1634483E+08]
     }
 ],
```


"Isotropic" or "Orthotropic"

JSON specification file for ANCF tire (2/2)

```
"Structural Damping Coefficient":  0.005,

"Default Pressure":                200.0e3,

"Bead Section":
{
    "Layer Thickness":   [ 0.5e-03 , 0.5e-02 , 0.5e-03 ],
    "Ply Angle":         [ 90 , 0 , 90],
    "Material ID":       [ 0 , 2 , 0],
    "Number Elements":   2
},

"Sidewall Section":
{
    "Layer Thickness":   [ 0.5e-03 , 0.1e-03 , 0.5e-03 ],
    "Ply Angle":         [ 90 , 0 , 90 ],
    "Material ID":       [ 0 , 2 , 0 ],
    "Number Elements":   4
},

"Tread Section":
{
    "Layer Thickness":   [ 0.1e-02 , 0.3e-03 , 0.3e-03 , 0.5e-03 ],
    "Ply Angle":         [ 0 , -20 , 20 , 90 ],
    "Material ID":       [ 2 , 1 , 1 , 0 ],
    "Number Elements":   6
},
```

```
"Number Elements Circumference":  90,

 "Profile":
 [
     [ 0.000000E+00 , 0.000000E+00 , -1.150000E-01 ],
     [ 1.428571E-02 , 1.166670E-02 , -1.164180E-01 ],
     [ 2.857143E-02 , 2.333330E-02 , -1.192300E-01 ],
     [ 4.285714E-02 , 3.500000E-02 , -1.230200E-01 ],
     [ 5.714286E-02 , 4.666670E-02 , -1.273710E-01 ],
     [ 7.142857E-02 , 5.833330E-02 , -1.318700E-01 ],
     [ 8.571429E-02 , 7.000000E-02 , -1.361330E-01 ],
     [ 1.000000E-01 , 8.166670E-02 , -1.399910E-01 ],
     [ 1.142857E-01 , 9.333330E-02 , -1.433510E-01 ],
     [ 1.285714E-01 , 1.050000E-01 , -1.461240E-01 ],
     [ 1.428571E-01 , 1.166670E-01 , -1.482160E-01 ],
     [ 1.571429E-01 , 1.283330E-01 , -1.495390E-01 ],
     [ 1.714286E-01 , 1.400000E-01 , -1.500000E-01 ],

     .
     .
     .

     [ 9.571429E-01 , 3.500000E-02 ,  1.230200E-01 ],
     [ 9.714286E-01 , 2.333330E-02 ,  1.192300E-01 ],
     [ 9.857143E-01 , 1.166670E-02 ,  1.164180E-01 ],
     [ 1.000000E+00 , 0.000000E+00 ,  1.150000E-01 ]
 ]
}
```

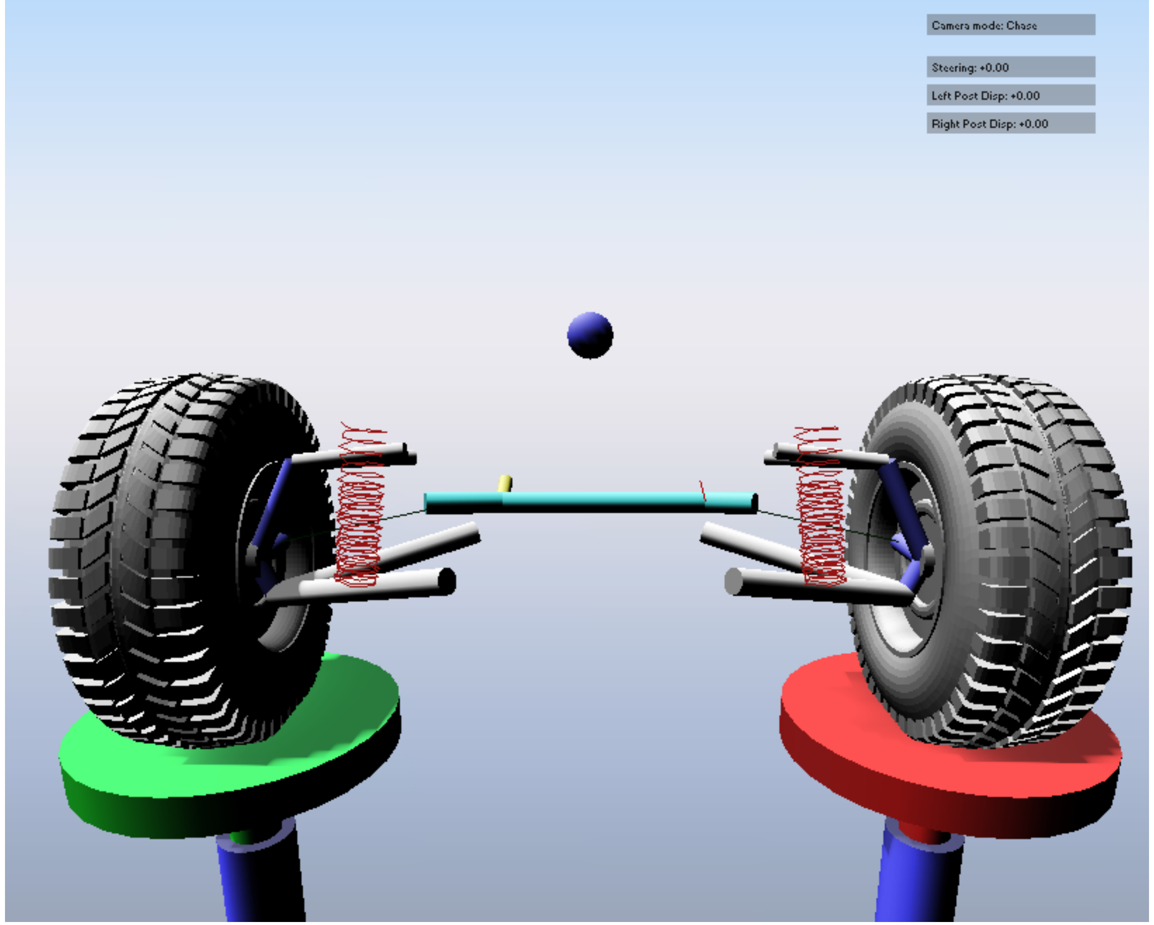
Parameterized 2D curve (→ "Profile")

α | **x(α)** | **y(α)** (columns of the profile entries)

Suspension Test Rig

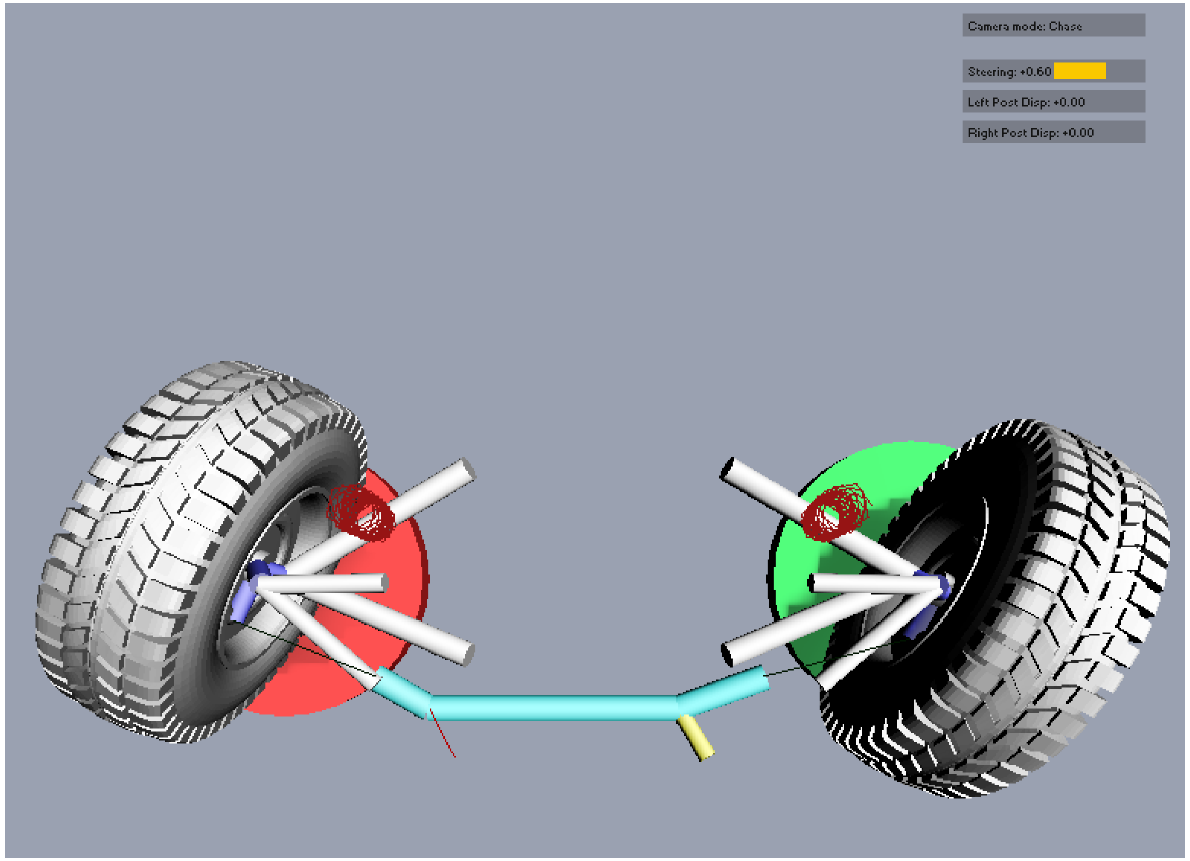
Suspension Test Rig

- Purpose:
  Analyze combined front suspension/steering subsystem behavior without involving complex full vehicle dynamics.
- Examples:
  1. Shock spring prelength/preload
  2. Steering wheel input limits



Suspension Test Rig Overview

- Similar to a full vehicle system
- Front half, chassis fixed to ground
- Shaker posts added to the system
    - Left=Green, Right=Red
- Constrained to move only vertically (global)
- Linear Actuator specifies post displacement
- Wheel spindle body CM point constrained in plane
    - Plane vertically offset from shaker post surface
- Logs important measurements to console or file



JSON specification file for SuspensionTestRig

```json
{
  "Name":                        "HMMWV Front Suspension Test",
  "Type":                        "SuspensionTest",
  "Template":                    "SuspensionTest",

  "Suspension":
  {
    "Input File":                "hmmwv/suspension/HMMWV_DoubleWishboneFront.json",
    "Location":                 [1.688965, 0, 0],
    "Left Wheel Input File":     "hmmwv/wheel/HMMWV_Wheel_FrontLeft.json",
    "Right Wheel Input File":    "hmmwv/wheel/HMMWV_Wheel_FrontRight.json"
  },

  "Steering":
  {
    "Input File":                "hmmwv/steering/HMMWV_PitmanArm.json",
    "Location":                  [1.24498, 0, 0.101322],
    "Orientation":               [0.98699637, 0, 0.16074256, 0],
    "Suspension Index":          0
  }
}
```

Subsystem type (string)

Template type (string)

Subsystems (suspension and steering) specified as in a wheeled vehicle JSON specification file. The block specifying the steering subsystem is optional.

Suspension Subsystem

ChSuspension base class

- Defines the common interface for any suspension subsystem
- All classes defining particular suspension templates inherit from ChSuspension

```
///
/// Base class for a suspension subsystem.
///
class CH_VEHICLE_API ChSuspension : public ChPart
```

ChSuspension base class members

- A ChSuspension has:

```
std::shared_ptr<ChBody> m_spindle[2];                  ///< handles to spindle bodies
std::shared_ptr<ChShaft> m_axle[2];                    ///< handles to axle shafts
std::shared_ptr<ChShaftsBody> m_axle_to_spindle[2];    ///< handles to spindle-shaft connectors
std::shared_ptr<ChLinkLockRevolute> m_revolute[2];     ///< handles to spindle revolute joints
```

ChSuspension base class accessors

- A ChSuspension provides access to:
  - Its constituent parts (spindle body, axle shaft, etc.)
  - States of the wheel (spindle) bodies
  - Angular speed of the axle shafts

ChSuspension base class methods

- A ChSuspension provides methods to:

```
/// Apply the provided tire forces.
/// The given tire force and moment is applied to the specified (left or
/// right) spindle body.  This function provides the interface to the tire
/// system (intermediated by the vehicle system).
void ApplyTireForce(
  ChVehicleSide side,            ///< spindle body (left or right) where forces should be applied
  const ChTireForce& tire_force  ///< generalized tire forces
  );

/// Apply the provided motor torque.
/// The given torque is applied to the specified (left or right) axle. This
/// function provides the interface to the drivetrain subsystem (intermediated
/// by the vehicle system).
void ApplyAxleTorque(
  ChVehicleSide side,   ///< indicates the axle (left or right) where the torque should be applied
  double torque         ///< value of applied torque
  );
```

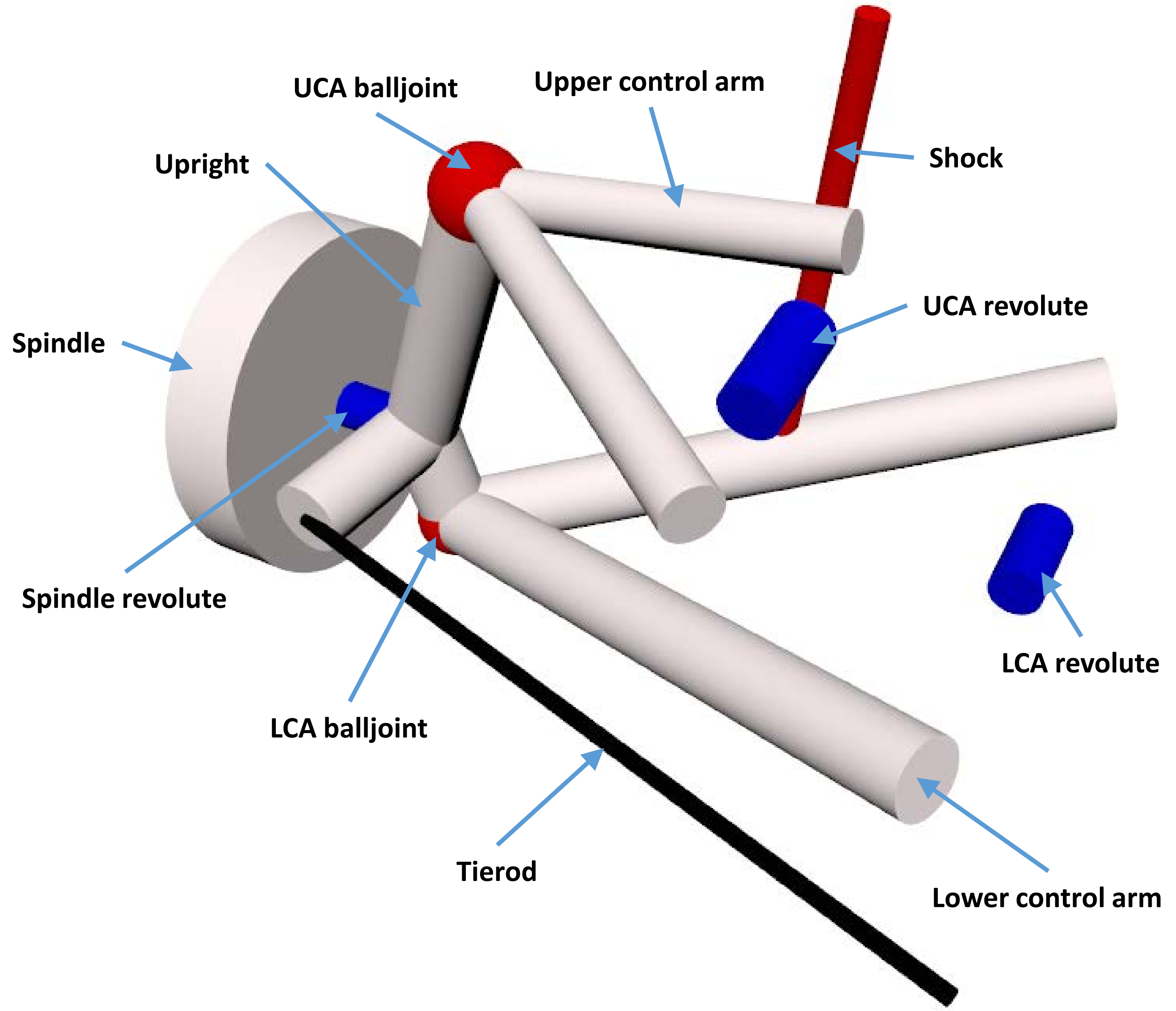
ChSuspension base class virtual methods

- A concrete class must implement

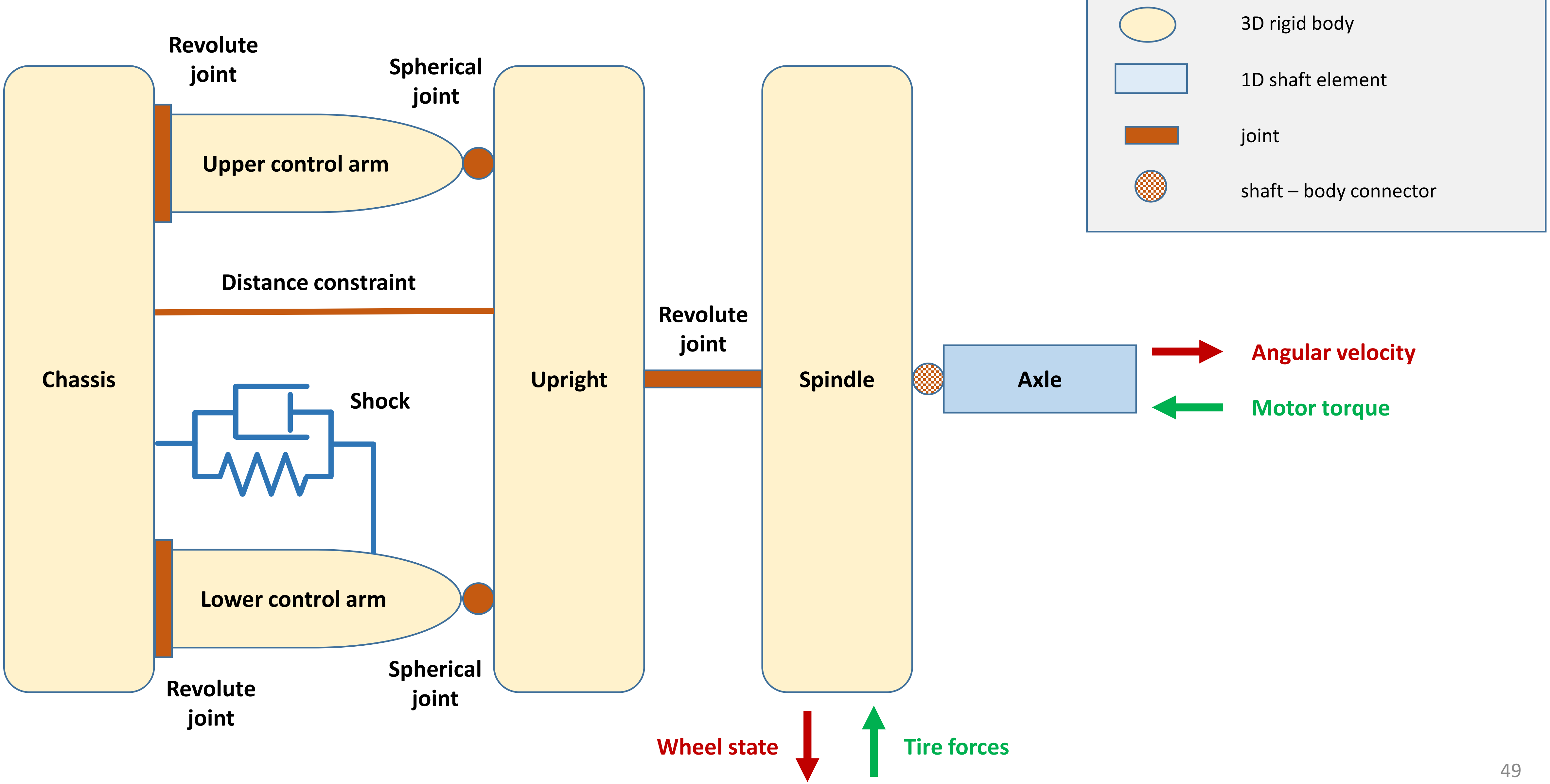
```
/// Initialize this suspension subsystem.
/// The suspension subsystem is initialized by attaching it to the specified
/// chassis body at the specified location (with respect to and expressed in
/// the reference frame of the chassis). It is assumed that the suspension
/// reference frame is always aligned with the chassis reference frame.
/// Finally, tierod_body is a handle to the body to which the suspension
/// tierods are to be attached. For a steerable suspension, this will be the
/// steering link of a suspension subsystem.  Otherwise, this is the chassis.
virtual void Initialize(std::shared_ptr<ChBodyAuxRef> chassis,  ///< [in] handle to the chassis body
                        const ChVector<>& location,             ///< [in] location relative to the chassis frame
                        std::shared_ptr<ChBody> tierod_body,    ///< [in] body to which tierods are connected
                        double left_ang_vel = 0,                ///< [in] initial angular velocity of left wheel
                        double right_ang_vel = 0                ///< [in] initial angular velocity of right wheel
                        ) = 0;
```

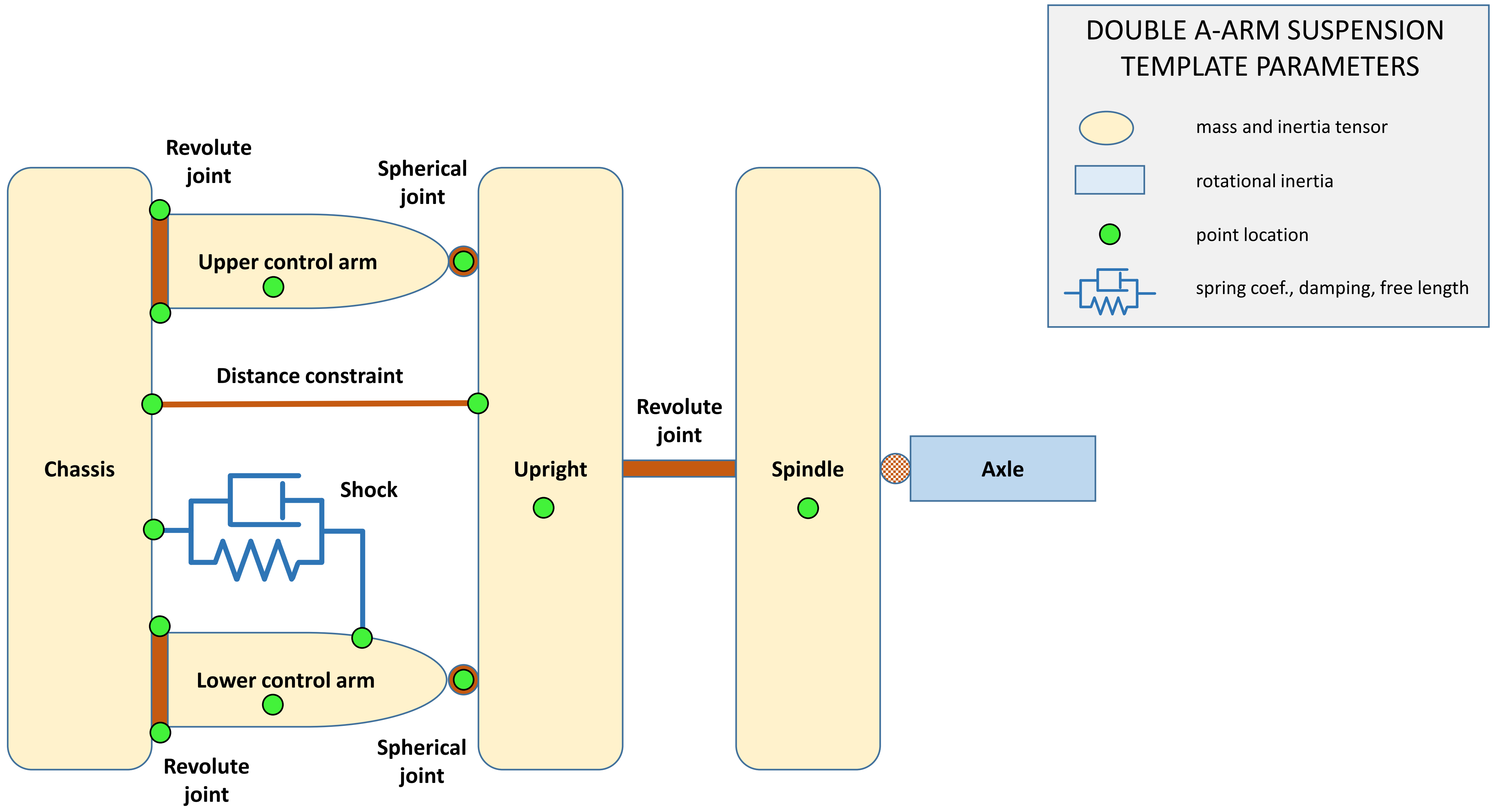
Suspension Templates

Double Wishbone (full and reduced)

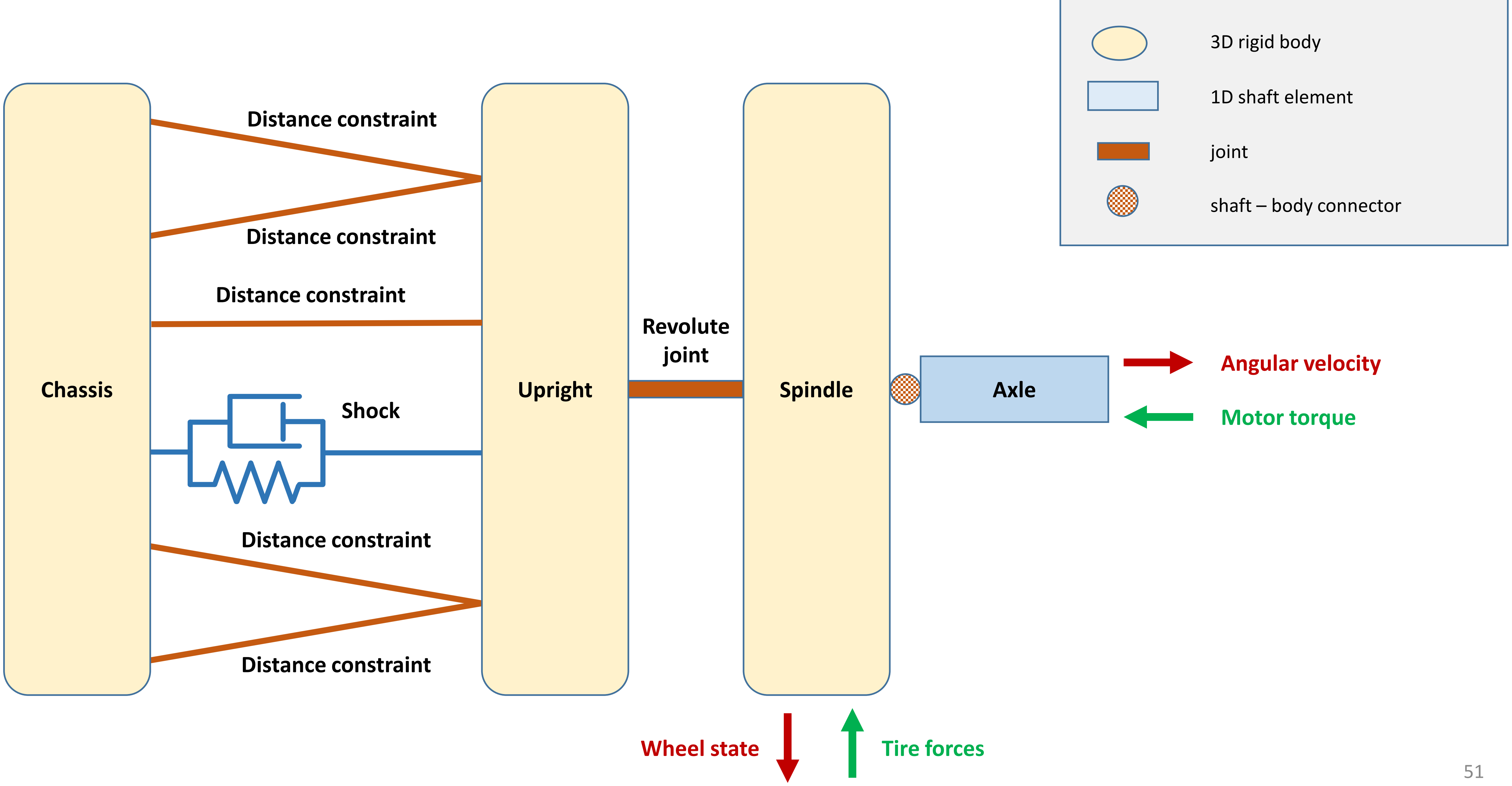

UCA balljoint
Upper control arm
Upright
Shock
UCA revolute
Spindle
Spindle revolute
LCA revolute
LCA balljoint
Tierod
Lower control arm

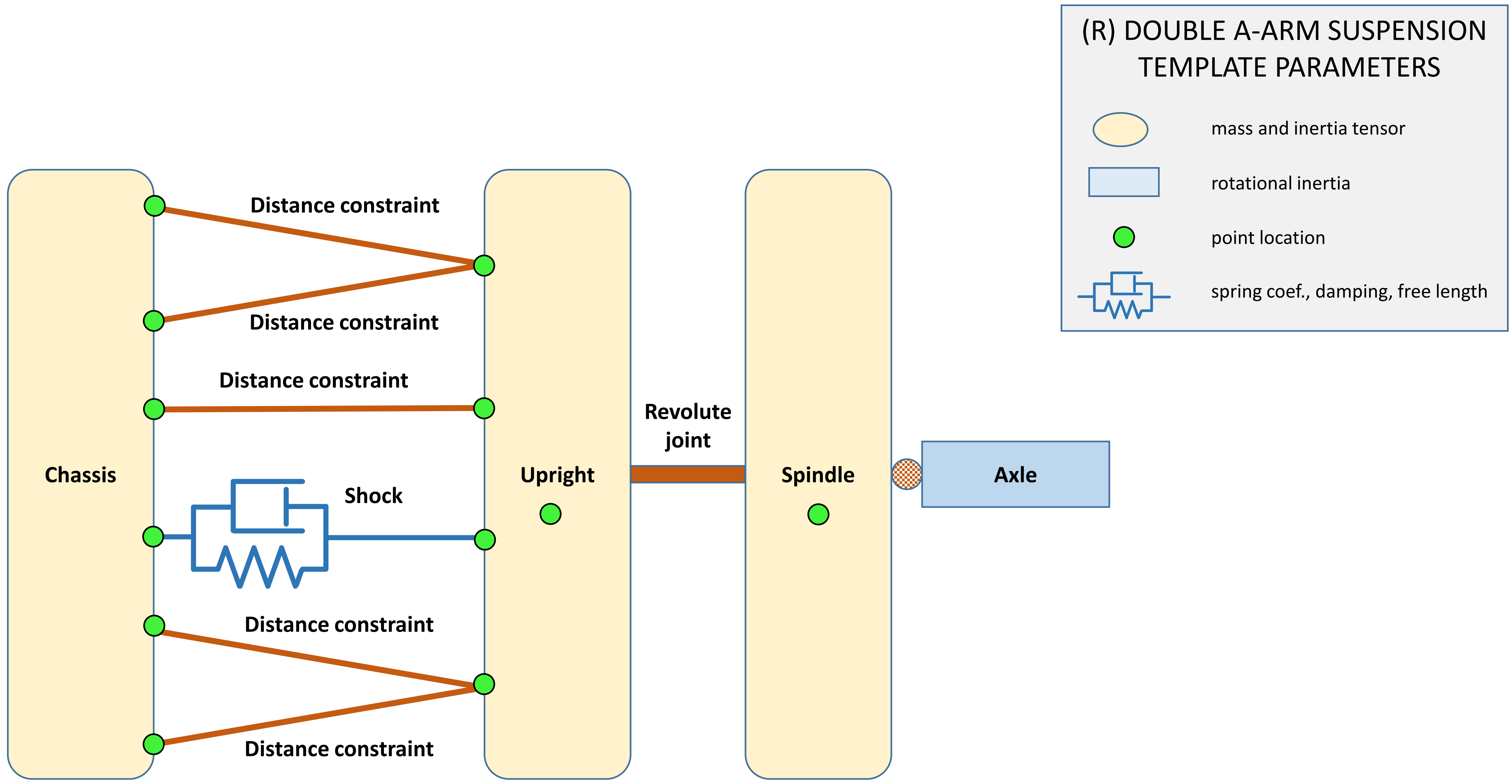
DOUBLE A-ARM SUSPENSION TEMPLATE

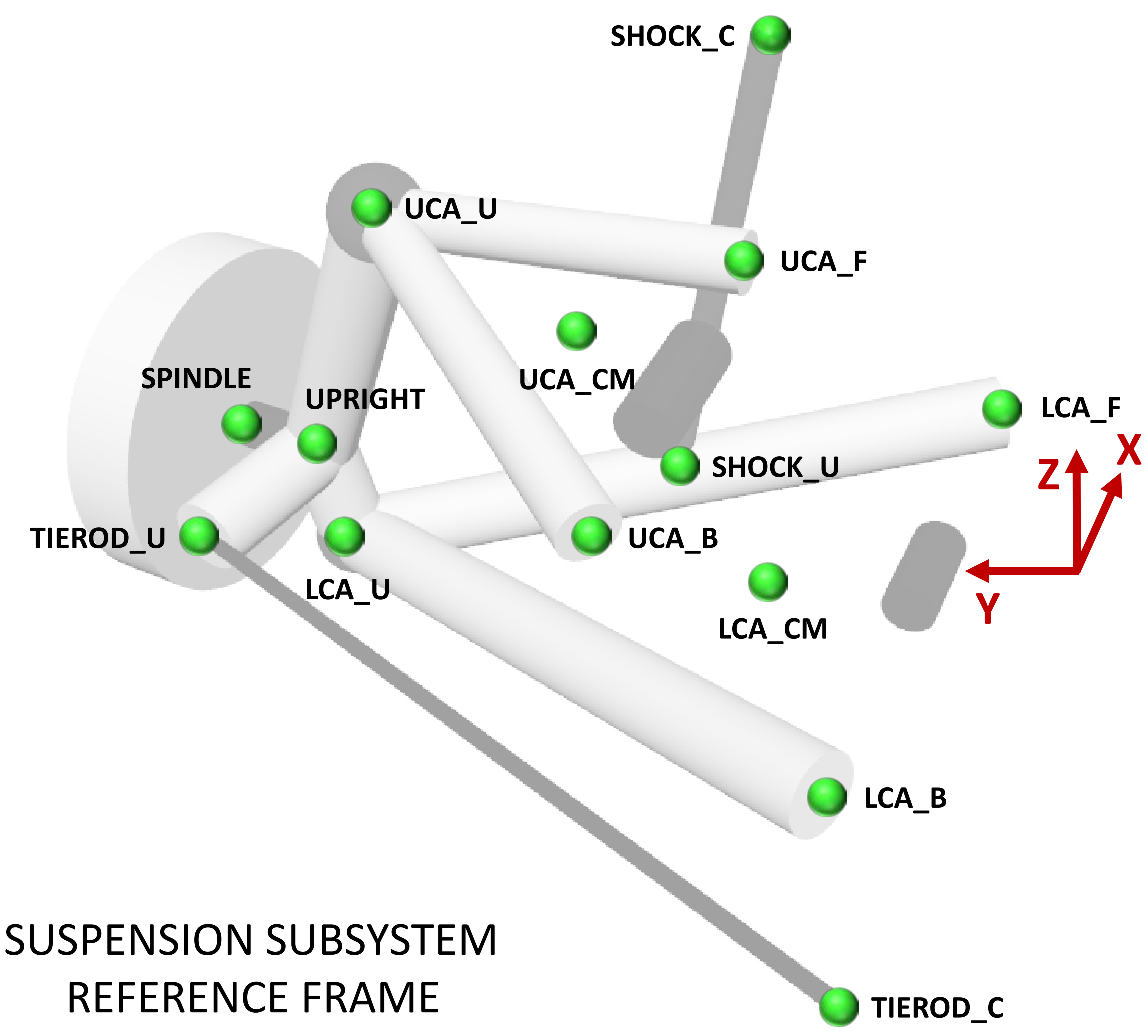



DOUBLE A-ARM SUSPENSION
TEMPLATE PARAMETERS
mass and inertia tensor
rotational inertia
point location
spring coef., damping, free length
Revolute
joint
Spherical
joint
Upper control arm
Distance constraint
Chassis
Shock
Upright
Revolute
joint
Spindle
Axle
Lower control arm
Revolute
joint
Spherical
joint

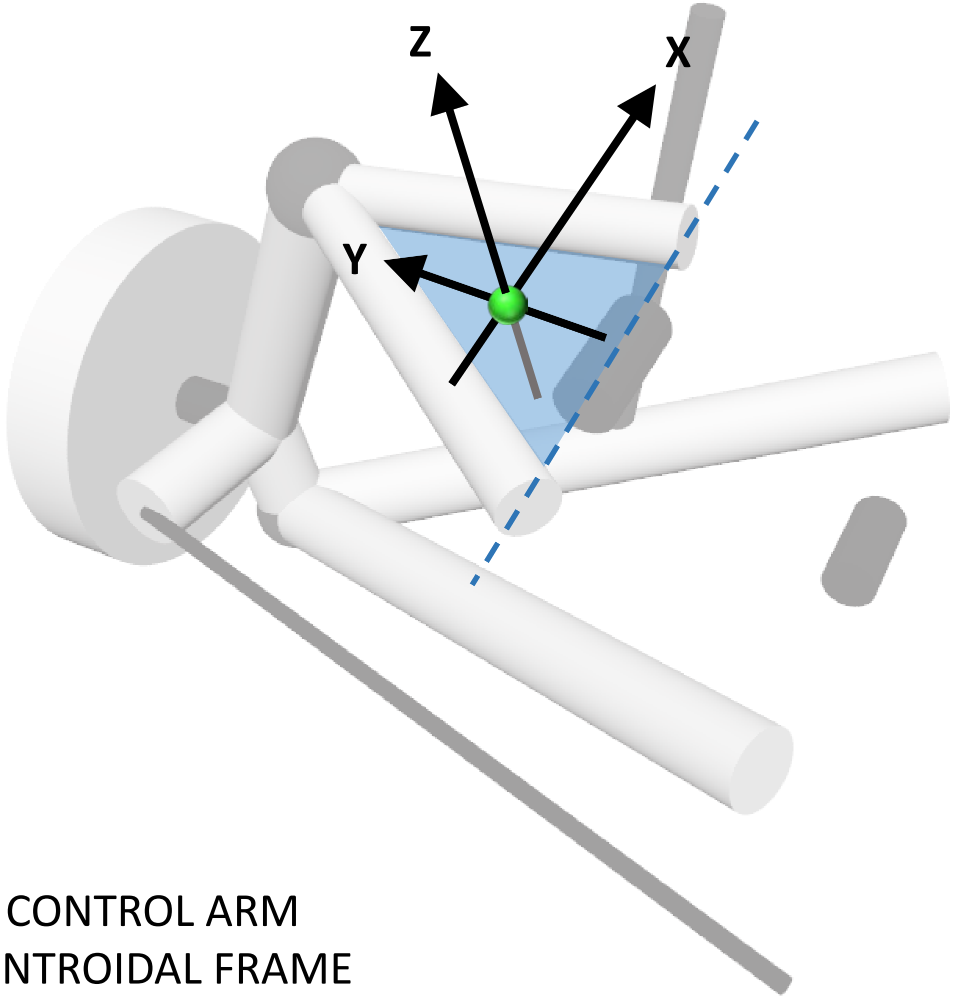
(R) DOUBLE A-ARM SUSPENSION TEMPLATE




(R) DOUBLE A-ARM SUSPENSION TEMPLATE PARAMETERS
mass and inertia tensor
rotational inertia
point location
spring coef., damping, free length
Chassis
Distance constraint
Distance constraint
Distance constraint
Shock
Distance constraint
Distance constraint
Upright
Revolute joint
Spindle
Axle



SUSPENSION SUBSYSTEM
REFERENCE FRAME



CONTROL ARM
CENTROIDAL FRAME

ChDoubleWishbone base class

```
///
/// Base class for a double-A arm suspension modeled with bodies and constraints.
/// Derived from ChSuspension, but still an abstract base class.
///
/// The suspension subsystem is modeled with respect to a right-handed frame,
/// with X pointing towards the front, Y to the left, and Z up (ISO standard).
/// The suspension reference frame is assumed to be always aligned with that of
/// the vehicle.  When attached to a chassis, only an offset is provided.
///
/// All point locations are assumed to be given for the left half of the
/// suspension and will be mirrored (reflecting the y coordinates) to construct
/// the right side.
///
class CH_VEHICLE_API ChDoubleWishbone : public ChSuspension
```

ChDoubleWishbone (pure) virtual functions

```
/// Identifiers for the various hardpoints.
enum PointId {
  SPINDLE,    ///< spindle location
  UPRIGHT,    ///< upright location
  UCA_F,      ///< upper control arm, chassis front
  UCA_B,      ///< upper control arm, chassis back
  UCA_U,      ///< upper control arm, upright
  UCA_CM,     ///< upper control arm, center of mass
  LCA_F,      ///< lower control arm, chassis front
  LCA_B,      ///< lower control arm, chassis back
  LCA_U,      ///< lower control arm, upright
  LCA_CM,     ///< lower control arm, center of mass
  SHOCK_C,    ///< shock, chassis
  SHOCK_A,    ///< shock, lower control arm
  SPRING_C,   ///< spring, chassis
  SPRING_A,   ///< spring, lower control arm
  TIEROD_C,   ///< tierod, chassis
  TIEROD_U,   ///< tierod, upright
  NUM_POINTS
};

/// Return the location of the specified hardpoint.
/// The returned location must be expressed in the suspension reference frame.
virtual const ChVector<> getLocation(PointId which) = 0;
```

ChDoubleWishbone (pure) virtual functions

```
/// Return the mass of the spindle body.
virtual double getSpindleMass() const = 0;
/// Return the mass of the upper control arm body.
virtual double getUCAMass() const = 0;
/// Return the mass of the lower control body.
virtual double getLCAMass() const = 0;
/// Return the mass of the upright body.
virtual double getUprightMass() const = 0;

/// Return the moments of inertia of the spindle body.
virtual const ChVector<>& getSpindleInertia() const = 0;
/// Return the moments of inertia of the upper control arm body.
virtual const ChVector<>& getUCAInertia() const = 0;
/// Return the moments of inertia of the lower control arm body.
virtual const ChVector<>& getLCAInertia() const = 0;
/// Return the moments of inertia of the upright body.
virtual const ChVector<>& getUprightInertia() const = 0;

/// Return the inertia of the axle shaft.
virtual double getAxleInertia() const = 0;
```

ChDoubleWishbone (pure) virtual functions

```
/// Indicate whether the spring is modeled as a nonlinear element.
/// If true, the concrete class must provide a callback function to calculate
/// the force in the spring element (see getSpringForceCallback).
virtual bool useNonlinearSpring() const { return false; }
/// Indicate whether the shock is modeled as a nonlinear element.
/// If true, the concrete class must provide a callback function to calculate
/// the force in the shock element (see getShockForceCallback).
virtual bool useNonlinearShock() const  { return false; }

/// Return the spring coefficient (for linear spring elements).
virtual double getSpringCoefficient() const  { return 1.0; }
/// Return the damping coefficient (for linear shock elements).
virtual double getDampingCoefficient() const { return 1.0; }

/// Return the free (rest) length of the spring element.
virtual double getSpringRestLength() const = 0;

/// Return the callback function for spring force (for nonlinear spring).
virtual ChSpringForceCallback* getSpringForceCallback() const { return NULL; }
/// Return the callback function for shock force (for nonlinear shock).
virtual ChSpringForceCallback* getShockForceCallback()  const { return NULL; }
```

ChDoubleWishbone (pure) virtual functions

```
/// Return the radius of the spindle body (visualization only).
virtual double getSpindleRadius() const = 0;
/// Return the width of the spindle body (visualization only).
virtual double getSpindleWidth() const = 0;
/// Return the radius of the upper control arm body (visualization only).
virtual double getUCARadius() const = 0;
/// Return the radius of the lower control arm body (visualization only).
virtual double getLCARadius() const = 0;
/// Return the radius of the upright body (visualization only).
virtual double getUprightRadius() const = 0;
```

JSON specification for double wishbone

```
{
  "Name":                         "Generic DoubleWishbone Front",
  "Type":                         "Suspension",
  "Template":                     "DoubleWishbone",

  "Spindle" :
  {
    "Mass":                       1.103,
    "COM":                        [-0.040, 0.910, -0.026],
    "Inertia":                    [0.000478, 0.000496, 0.000478],
    "Radius":                     0.15,
    "Width":                      0.06
  },

  "Upright":
  {
    "Mass":                       1.397,
    "COM":                        [-0.040, 0.910, -0.026],
    "Inertia":                    [0.0138, 0.0146, 0.00283],
    "Radius":                     0.025
  },
```

- "Name": Identifier name (string)
- "Type": Subsystem type (string)
- "Template": Template type (string)
- "Mass": Mass in kg (float)
- "COM": Location of the center of mass in m (float) with respect to the suspension reference frame
- "Inertia": Moments of inertia in kg m^2 (float)
- "Radius" and "Width": Visualization dimensions in m (float)

JSON specification for double wishbone

```
"Upper Control Arm":
 {
   "Mass":                     1.032,
   "COM":                      [-0.196, 0.645, 0.245],
   "Inertia":                  [0.00591, 0.00190, 0.00769],
   "Radius":                   0.02,
   "Location Chassis Front":   [-0.160, 0.539, 0.243],
   "Location Chassis Back":    [-0.339, 0.587, 0.249],
   "Location Upright":         [-0.088, 0.808, 0.243]
 },

 "Lower Control Arm":
 {
   "Mass":                     1.611,
   "COM":                      [-0.040, 0.639, -0.224],
   "Inertia":                  [0.0151, 0.0207, 0.0355],
   "Radius":                   0.03,
   "Location Chassis Front":   [0.199, 0.479, -0.206],
   "Location Chassis Back":    [-0.279, 0.539, -0.200],
   "Location Upright":         [-0.040, 0.898, -0.265]
 },
```

Locations of the front and back attachment points between the control arm and the chassis in m (float) with respect to the suspension reference frame. These points form the axis of rotation for the revolute joint between the chassis and the control arm

Location of the spherical joint between the control arm and the upright in m (float) with respect to the suspension reference frame

JSON specification for double wishbone

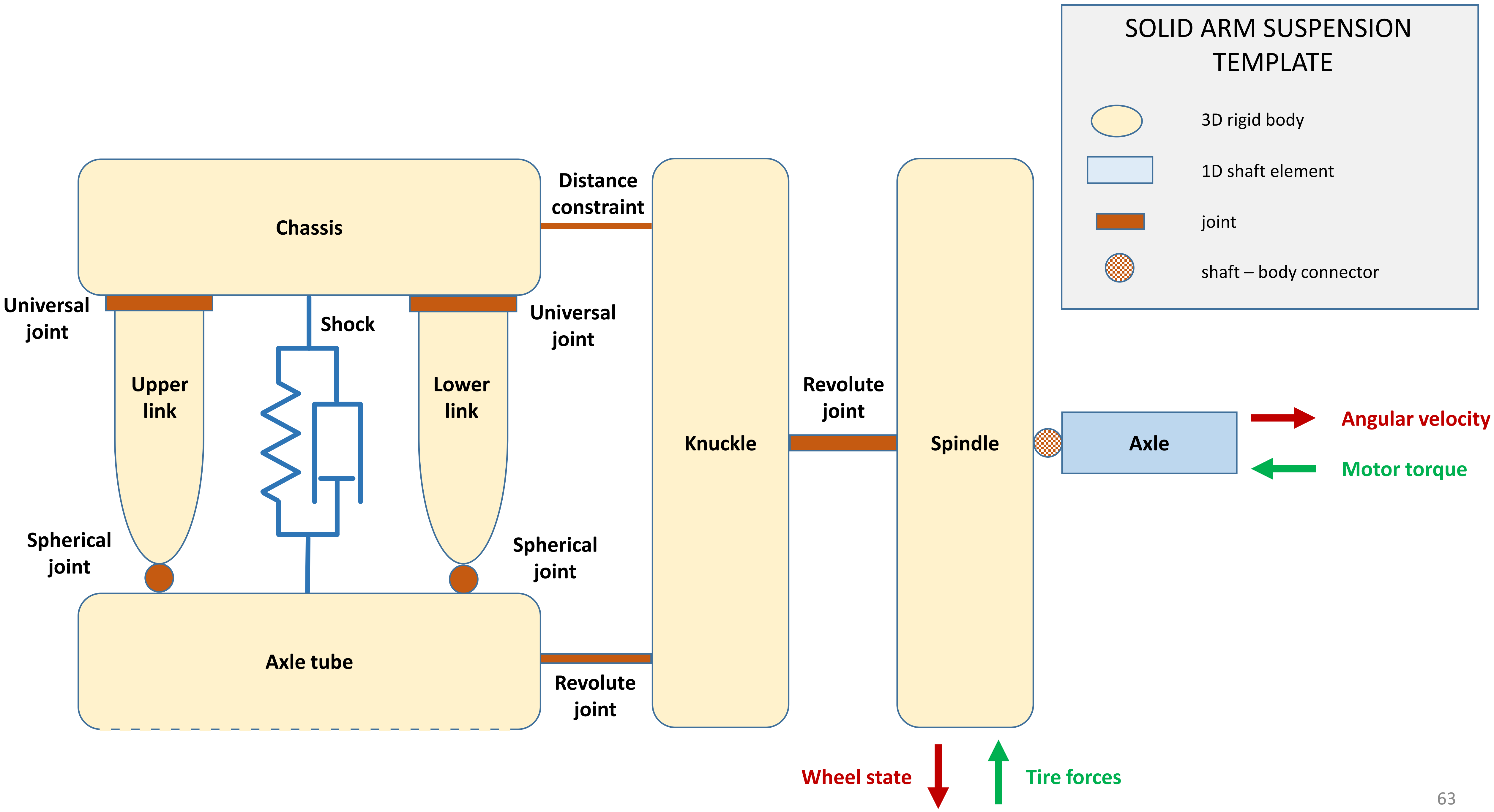
```
 "Tierod":
  {
    "Location Chassis":         [-0.279, 0.479, -0.026],
    "Location Upright":         [-0.220, 0.898, -0.026]
  },

  "Spring":
  {
    "Location Chassis":         [-0.064, 0.659, 0.094],
    "Location Arm":             [-0.040, 0.718, -0.206],
    "Spring Coefficient":       369149.000,
    "Free Length" :             0.356
  },

  "Shock":
  {
    "Location Chassis":         [-0.088, 0.599, 0.393],
    "Location Arm":             [-0.040, 0.718, -0.206],
    "Damping Coefficient":      22459.000
  },

  "Axle":
  {
    "Inertia":                  0.4
  }
}
```

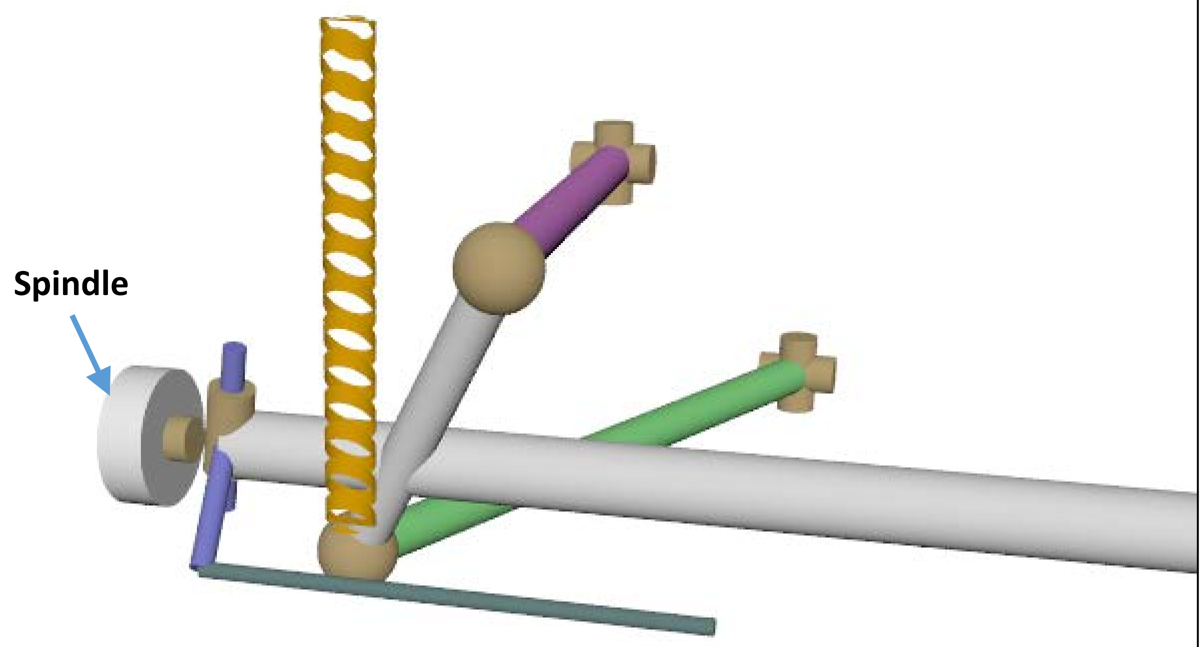
Locations of the upper and lower attachment points between the control arm and the chassis in m (float) with respect to the suspension reference frame

Rotational moment of inertia in kg m^2 (float)

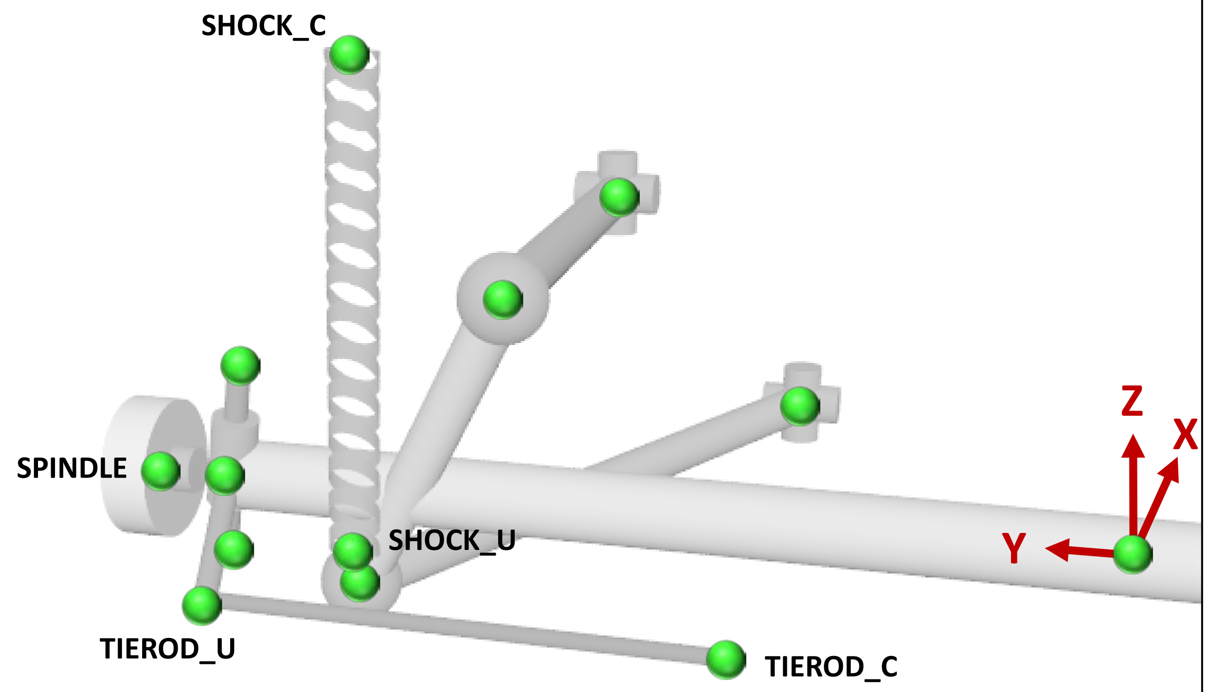
Suspension Templates

Solid Axle


SOLID ARM SUSPENSION TEMPLATE
3D rigid body
1D shaft element
joint
shaft – body connector
Chassis
Distance constraint
Universal joint
Shock
Universal joint
Upper link
Lower link
Revolute joint
Knuckle
Spindle
Axle
Angular velocity
Motor torque
Spherical joint
Spherical joint
Axle tube
Revolute joint
Wheel state
Tire forces



SUSPENSION SUBSYSTEM
PARTS



SUSPENSION SUBSYSTEM
HARDPOINTS

JSON specification file for solid axle

```
{
  "Name":                        "Generic Solid Axle Front",
  "Type":                        "Suspension",
  "Template":                    "SolidAxle",

  "Spindle" :
  {
    "Mass":                      0.248,
    "COM":                       [0, 0.910, 0],
    "Inertia":                   [0.0000558, 0.0000279, 0.0000558],
    "Radius":                    0.06,
    "Width":                     0.04
  },

  "Knuckle":
  {
    "Mass":                      1.356,
    "COM":                       [-0.006, 0.834, 0.015],
    "Inertia":                   [0.00255, 0.00134, 0.00196],
    "Radius":                    0.01,
    "Location Lower":            [0.006, 0.849, -0.061],
    "Location Upper":            [-0.018, 0.819, 0.091]
  },
```

- Identifier name (string)
- Subsystem type (string)
- Template type (string)
- Mass in kg (float)
- Location of the center of mass in m (float) with respect to the suspension reference frame
- Moments of inertia in kg m^2 (float)
- Visualization dimensions in m (float)
- Locations of the upper and lower attachment points between the knuckle and axle tube in m (float) with respect to the suspension reference frame. These points form the axis of rotation for the revolute joint between the knuckle and the axle tube

JSON specification file for solid axle

```
"Upper Link":
 {
   "Mass":                           1.446,
   "COM":                            [0.182, 0.591, 0.182],
   "Inertia":                        [0.011, 0.011, 0.000142],
   "Radius":                         0.02,
   "Location Axle":                  [-0.067, 0.576, 0.182],
   "Location Chassis":               [0.431, 0.606, 0.182],
   "Universal Joint Axis Link":      [0, -1, 0],
   "Universal Joint Axis Chassis":   [0, 0, 1]
 },
 "Lower Link":
 {
   "Mass":                           2.892,
   "COM":                            [0.279, 0.577, -0.073],
   "Inertia":                        [0.0514, 0.0514, 0.00037],
   "Radius":                         0.02,
   "Location Axle":                  [0.012, 0.728, -0.091],
   "Location Chassis":               [0.546, 0.425, -0.055],
   "Universal Joint Axis Link":      [0, 1, 0],
   "Universal Joint Axis Chassis":   [0, 0, 1]
 },
 "Axle Tube" :
 {
   "Mass":                           44.958,
   "COM":                            [0, 0, 0],
   "Inertia":                        [7.744, 0.045, 7.744],
   "Radius":                         0.03
 },
```

Location of the spherical joint that attaches the link to the axle tube with respect to the suspension reference frame in m (float)

Location of the universal joint that attaches the link to the axle tube with respect to the suspension reference frame in m (float)

Directions of the universal joint that attaches the link to the axle tube

JSON specification file for solid axle

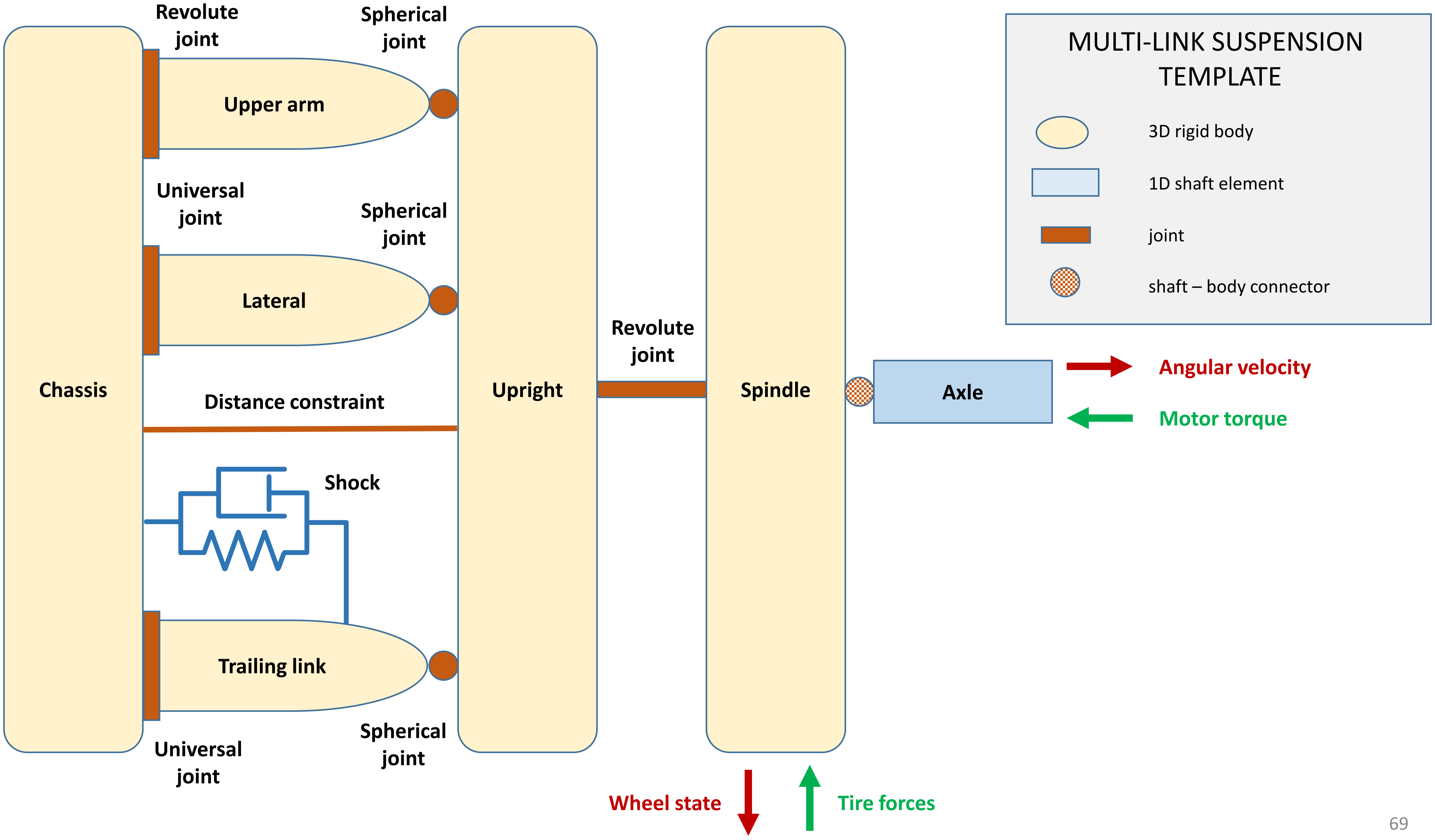
```
 "Tierod":
  {
    "Location Chassis":             [-0.091, 0.400, -0.079],
    "Location Knuckle":             [-0.091, 0.825, -0.079]
  },

  "Spring":
  {
    "Location Chassis":             [-0.097, 0.679, 0.364],
    "Location Axle":                [-0.079, 0.697, -0.030],
    "Spring Coefficient":           267062.000,
    "Free Length":                  0.3948
  },

  "Shock":
  {
    "Location Chassis":             [-0.097, 0.679, 0.364],
    "Location Axle":                [-0.079, 0.697, -0.030],
    "Damping Coefficient":          22459.000
  },

  "Axle":
  {
    "Inertia":                      0.4
  }
}
```

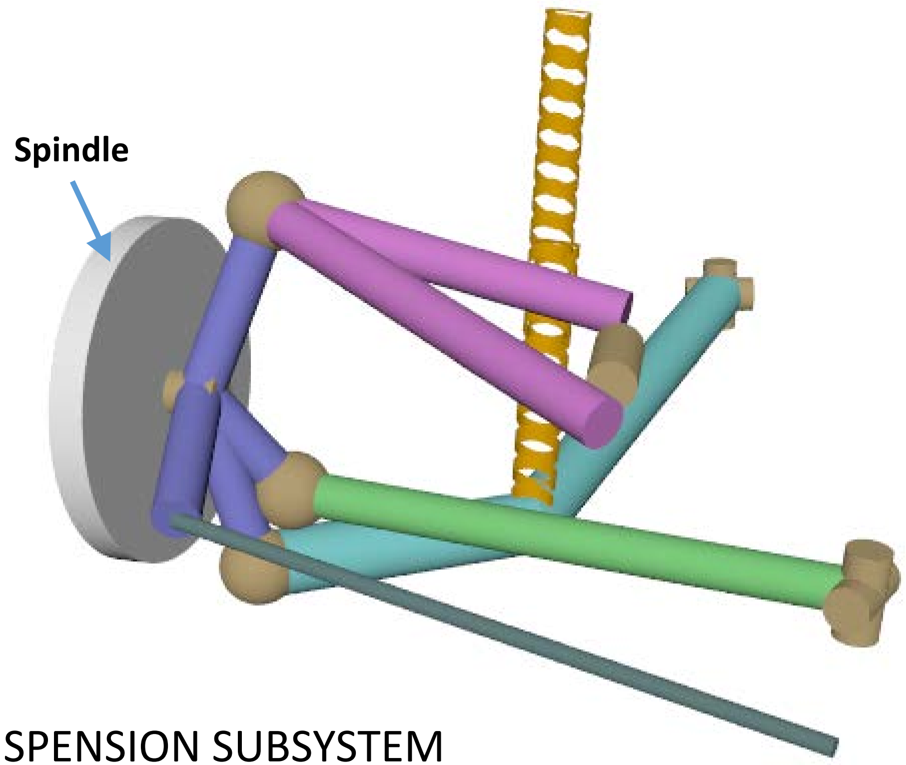
Locations of the upper and lower attachment points between the control arm and the chassis in m (float) with respect to the suspension reference frame

Rotational moment of inertia in kg m^2 (float)

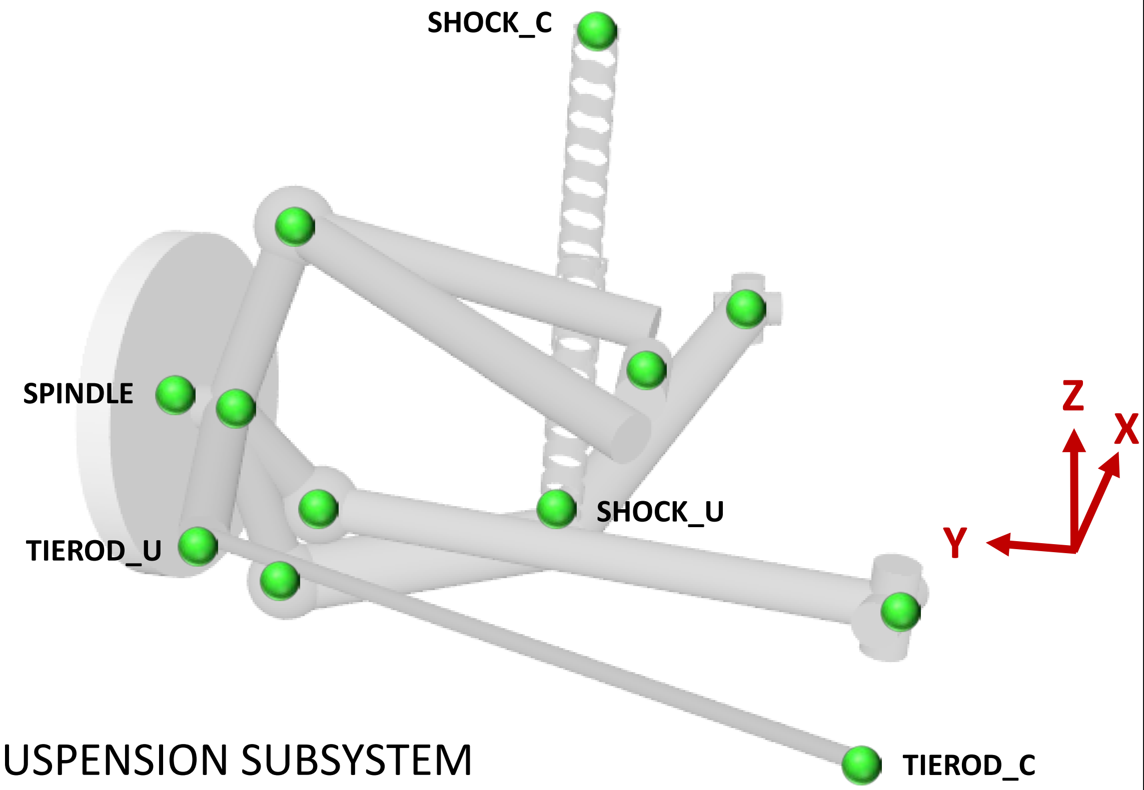
Suspension Templates

Multi-link


MULTI-LINK SUSPENSION TEMPLATE
3D rigid body
1D shaft element
joint
shaft – body connector
Revolute joint
Spherical joint
Upper arm
Universal joint
Spherical joint
Lateral
Chassis
Distance constraint
Upright
Revolute joint
Spindle
Axle
Angular velocity
Motor torque
Shock
Trailing link
Universal joint
Spherical joint
Wheel state
Tire forces



SUSPENSION SUBSYSTEM
PARTS



SUSPENSION SUBSYSTEM
HARDPOINTS

JSON specification file for multi-link

```
{
  "Name":                     "Generic Multi-Link Front",
  "Type":                     "Suspension",
  "Template":                 "MultiLink",

  "Spindle" :
  {
    "Mass":                   1.103,
    "COM":                    [0.000, 0.950, 0.000],
    "Inertia":                [0.000478, 0.000478, 0.000496],
    "Radius":                 0.15,
    "Width":                  0.03
  },

  "Upright":
  {
    "Mass":                   3.201,
    "COM":                    [0.000, 0.910, 0.000],
    "Inertia":                [0.0250, 0.00653, 0.0284],
    "Radius":                 0.02
  },
```

Identifier name (string)

Subsystem type (string)

Template type (string)

Mass in kg (float)

Location of the center of mass in m (float) with respect to the suspension reference frame

Moments of inertia in kg m^2 (float)

Visualization dimensions in m (float)

JSON specification file for multi-link


```
"Upper Arm":
 {
   "Mass":                          4.744,
   "COM":                           [-0.014, 0.640, 0.098],
   "Inertia":                       [0.0237, 0.0294, 0.00612],
   "Radius":                        0.02,
   "Location Chassis Front":        [0.060, 0.547, 0.082],
   "Location Chassis Back":         [-0.157, 0.508, 0.062],
   "Location Upright":              [0.056, 0.864, 0.151]
 },

 "Lateral":
 {
   "Mass":                          1.910,
   "COM":                           [0.033, 0.590, -0.113],
   "Inertia":                       [0.0543, 0.0541, 0.000279],
   "Radius":                        0.02,
   "Location Chassis":              [0.036, 0.338, -0.133],
   "Location Upright":              [0.029, 0.842, -0.093],
   "Universal Joint Axis Link":     [-0.978950, 0.204099, 0.0],
   "Universal Joint Axis Chassis": [-0.021990, -0.105472, 0.994179]
 },

 "Trailing Link":
 {
   "Mass":                          15.204,
   "COM":                           [0.279, 0.693, -0.132],
   "Inertia":                       [0.0762, 0.527, 0.567],
   "Radius":                        0.03,
   "Location Chassis":              [0.723, 0.599, -0.072],
   "Location Upright":              [-0.000, 0.864, -0.156],
   "Universal Joint Axis Link":     [0.0, 0.0, 1.0],
   "Universal Joint Axis Chassis": [-0.272, 0.962, 0.0]
 },
```

JSON specification file for multi-link

```
 "Tierod":
  {
    "Location Chassis":              [-0.257, 0.320, -0.116],
    "Location Upright":              [-0.144, 0.862, -0.056]
  },

  "Spring":
  {
    "Location Chassis":              [0.181, 0.641, 0.110],
    "Location Link":                 [0.181, 0.669, -0.164],
    "Spring Coefficient":            167062.000,
    "Free Length" :                  0.339
  },

  "Shock":
  {
    "Location Chassis":              [0.171, 0.628, 0.315],
    "Location Link":                 [0.181, 0.669, -0.162],
    "Damping Coefficient":           60068.000
  },

  "Axle":
  {
    "Inertia":                       0.166
  }
}
```

Support for non-linear spring-damper

- Currently supported by the ChDoubleWishbone and ChMultiLink suspension templates
- Implemented using the Chrono ChLinkSpringCB element which accepts a user-defined functor class

```
class ChSpringForceCallback {
public:
  virtual double operator()(
    double time,          ///< current time
    double rest_length,   ///< undeformed length
    double length,        ///< current length
    double vel            ///< current velocity (positive when extending)
    ) = 0;
};
```

- Default implementation is linear spring-damper
- A concrete, derived class can override the functor class to implement an arbitrary non-linear spring and/or damper
  - Example: HMMWV_DoubleWishboneFront & HMMWV_DoubleWishboneRear

Wheel Subsystem

Wheel subsystem

- A wheel subsystem does not own a body.
- It is just a container for mass properties and geometric information
- When attached to a suspension subsystem, the wheel's mass properties are used to update those of the spindle body owned by the suspension.
- A concrete wheel subsystem can optionally carry its own visualization assets (which are associated with the suspension's spindle body).

ChWheel base class

- Defines the common interface for a wheel subsystem

```
///
/// Base class for a vehicle wheel subsystem.
/// A wheel subsystem does not own a body. Instead, when attached to a suspension
/// subsystem, the wheel's mass properties are used to update those of the
/// spindle body owned by the suspension.
/// A concrete wheel subsystem can optionally carry its own visualization assets
/// (which are associated with the suspension's spindle body).
///
class CH_VEHICLE_API ChWheel : public ChPart
```

JSON specification file for wheel

```
{
  "Name":                          "HMMWV Wheel Front-Left",
  "Type":                          "Wheel",
  "Template":                      "Wheel",

  "Mass":                          88.39,
  "Inertia":                       [0.113, 0.113, 0.113],

  "Visualization":
  {
    "Mesh Filename":               "hmmwv/wheel_L.obj",
    "Mesh Name":                   "wheel_L_POV_geom",
    "Radius":                      0.41,
    "Width":                       0.254
  }
}
```

Subsystem type (string)

Template type (string)

Brake Subsystem

ChBrake base class

- Defines the common interface for any brake subsystem
- All classes defining particular brake templates inherit from ChBrake

```
///
/// Base class for a brake subsystem
///
class CH_VEHICLE_API ChBrake : public ChPart
```

Brake Templates

Simple brake

ChSimpleBrake

- Simple brake model using a constant torque opposing wheel rotation.
- Uses a speed-dependent torque
- It cannot simulate sticking
- On initialization, it is associated with a revolute joint connecting the spindle body
- Has a single parameter, the maximum braking torque

JSON specification file for SimpleBrake

```
{
  "Name":                          "HMMWV Brake Front",
  "Type":                          "Brake",
  "Template":                      "BrakeSimple",

  "Maximum Torque":                4000
}
```

Subsystem type (string)

Template type (string)

Maximum braking torque in Nm (double)

Steering Subsystem

ChSteering base class

- Defines the common interface for any steering subsystem
- All classes defining particular steering templates inherit from ChSteering

```
///
/// Base class for a steering subsystem.
///
class CH_VEHICLE_API ChSteering : public ChPart
```

ChSteering base class members

- A ChSteering has:

```
std::shared_ptr<ChBody> m_link;  ///< handle to the main steering link
```

ChSteering base class accessors

- A ChSteering provides access to:
  - Its constituent parts (the steering link body)

ChSteering base class virtual methods

- Initialize the steering subsystem relative to the chassis body

```
/// Initialize this steering subsystem.
/// The steering subsystem is initialized by attaching it to the specified
/// chassis body at the specified location (with respect to and expressed in
/// the reference frame of the chassis) and with specified orientation (with
/// respect to the chassis reference frame).
virtual void Initialize(std::shared_ptr<ChBodyAuxRef> chassis,  ///< [in] handle to the chassis body
                        const ChVector<>& location,             ///< [in] location relative to the chassis frame
                        const ChQuaternion<>& rotation          ///< [in] orientation relative to the chassis frame
                        ) = 0;
```

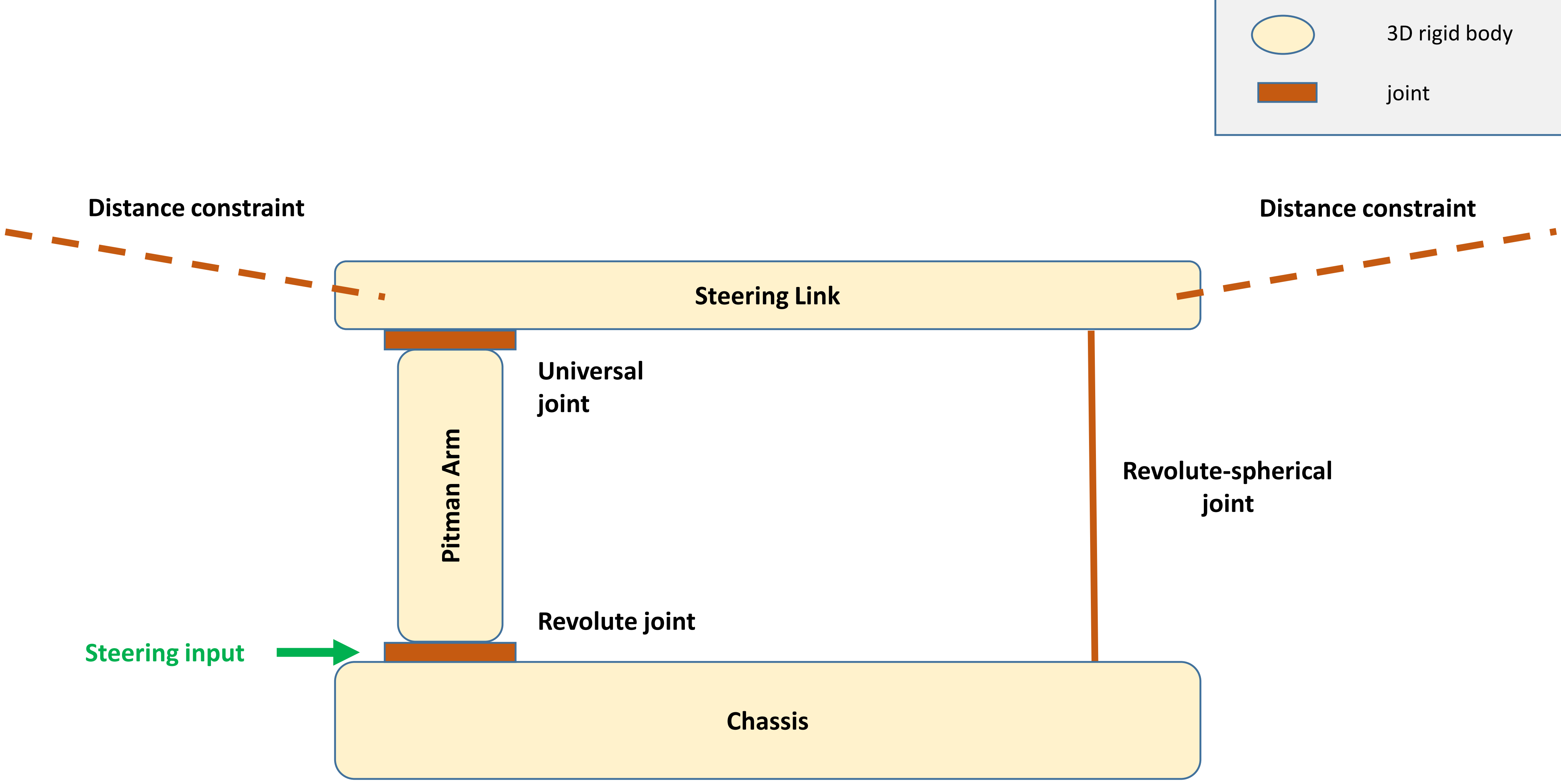
- Update the steering subsystem with data from the driver system

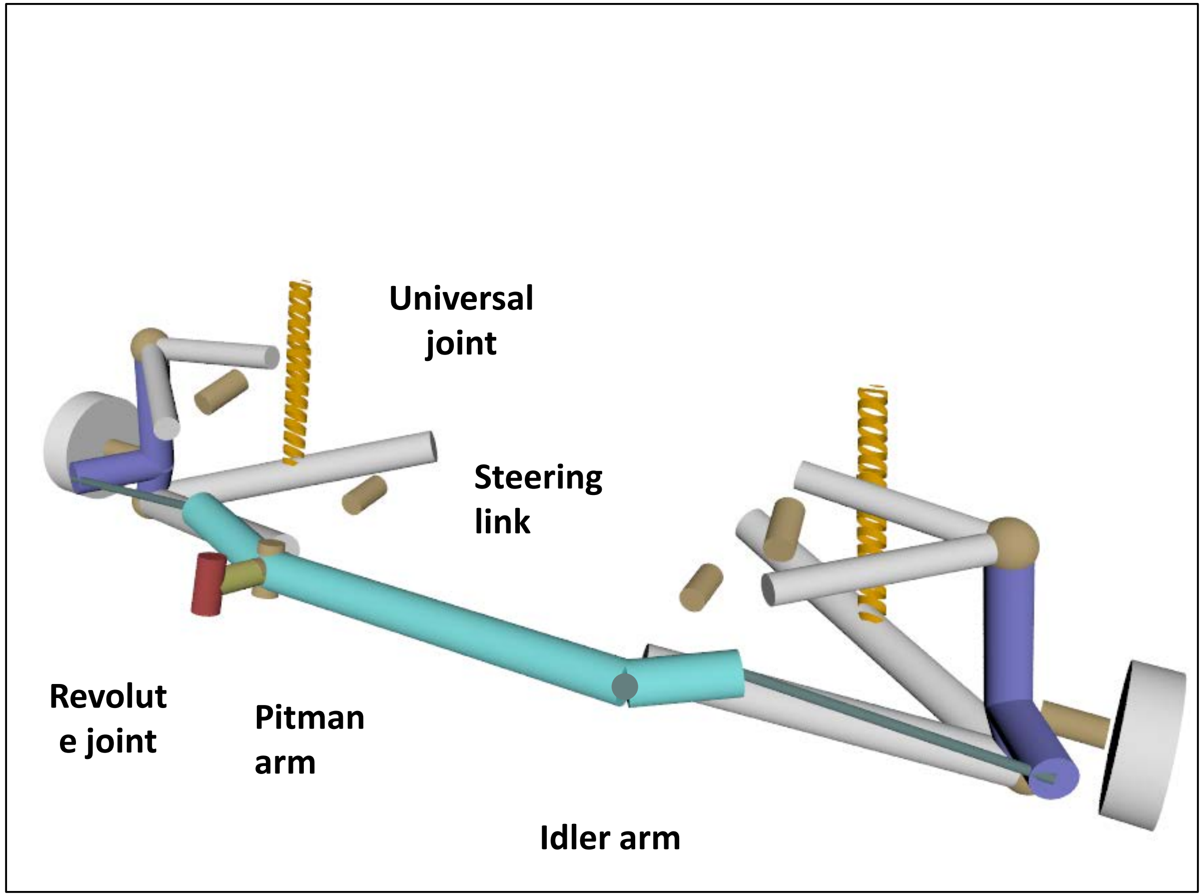
```
/// Update the state of this steering subsystem at the current time.
/// The steering subsystem is provided the current steering driver input (a
/// value between -1 and +1).  Positive steering input indicates steering
/// to the left. This function is called during the vehicle update.
virtual void Synchronize(double time,     ///< [in] current time
                         double steering  ///< [in] current steering input [-1,+1]
                         ) = 0;
```

Steering Templates

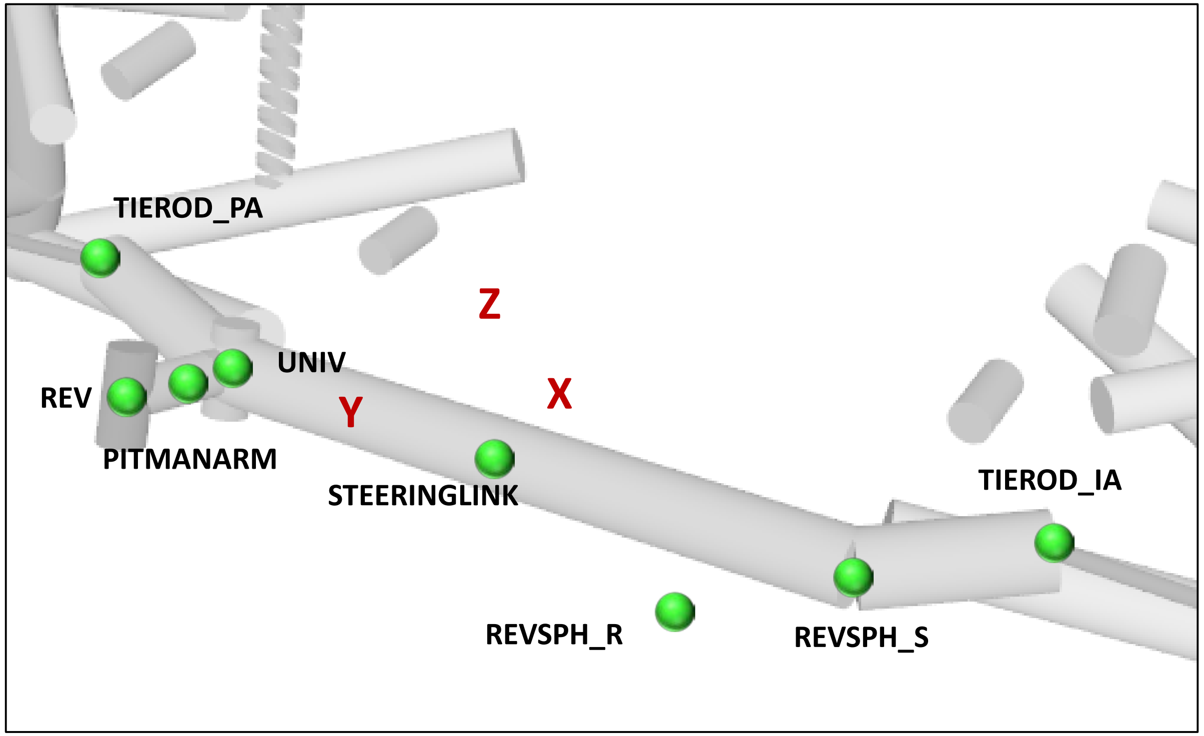
Pitman Arm

PITMAN ARM STEERING TEMPLATE





STEERING SUBSYSTEM
PARTS



STEERING SUBSYSTEM
HARDPOINTS

JSON specification for Pitman arm

```
{
  "Name":                          "HMMWV Pitman Arm Steering",
  "Type":                          "Steering",
  "Template":                      "PitmanArm",

  "Steering Link":
  {
    "Mass":                        9.072,
    "COM":                         [ 0.129, 0, 0],
    "Inertia":                     [1, 1, 1],
    "Radius":                      0.03
  },

  "Pitman Arm":
  {
    "Mass":                        2.259,
    "COM":                         [ 0.064, 0.249, 0],
    "Inertia":                     [1, 1, 1],
    "Radius":                      0.02
  },
```

Identifier name (string)

Subsystem type (string)

Template type (string)

JSON specification for Pitman arm

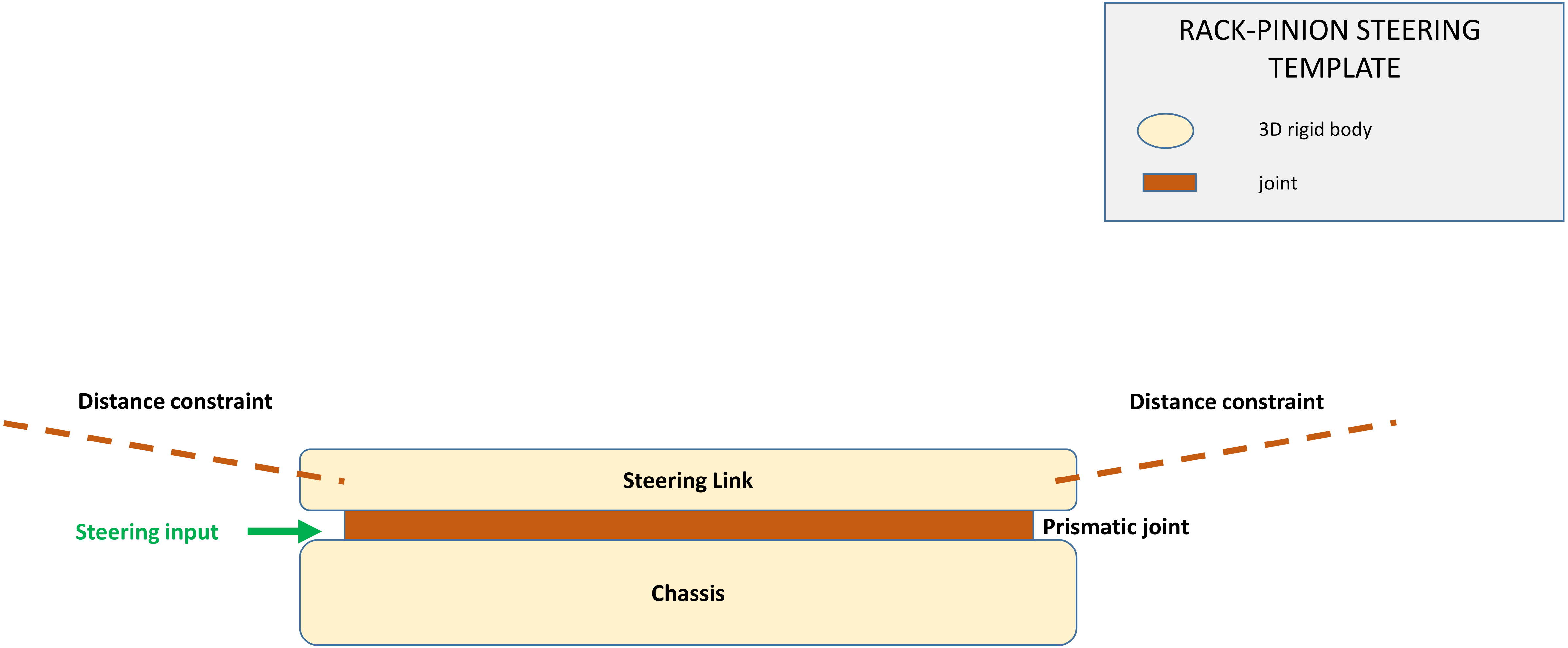
```
 "Revolute Joint":
  {
    "Location":                  [0, 0.249, 0],
    "Direction":                 [0, 0, 1],
    "Maximum Angle":             50
  },

  "Universal Joint":
  {
    "Location":                  [ 0.129, 0.249, 0],
    "Direction Arm":             [0, 0, 1],
    "Direction Link":            [1, 0, 0]
  },

  "Revolute-Spherical Joint":
  {
    "Location Chassis":          [0, -0.325, 0],
    "Location Link":             [0.129, -0.325, 0],
    "Direction":                 [0, 0, 1]
  },

  "Tierod Locations":
  {
    "Pitman Side":               [0.195,  0.448, 0.035],
    "Idler Side":                [0.195, -0.448, 0.035]
  }
}
```

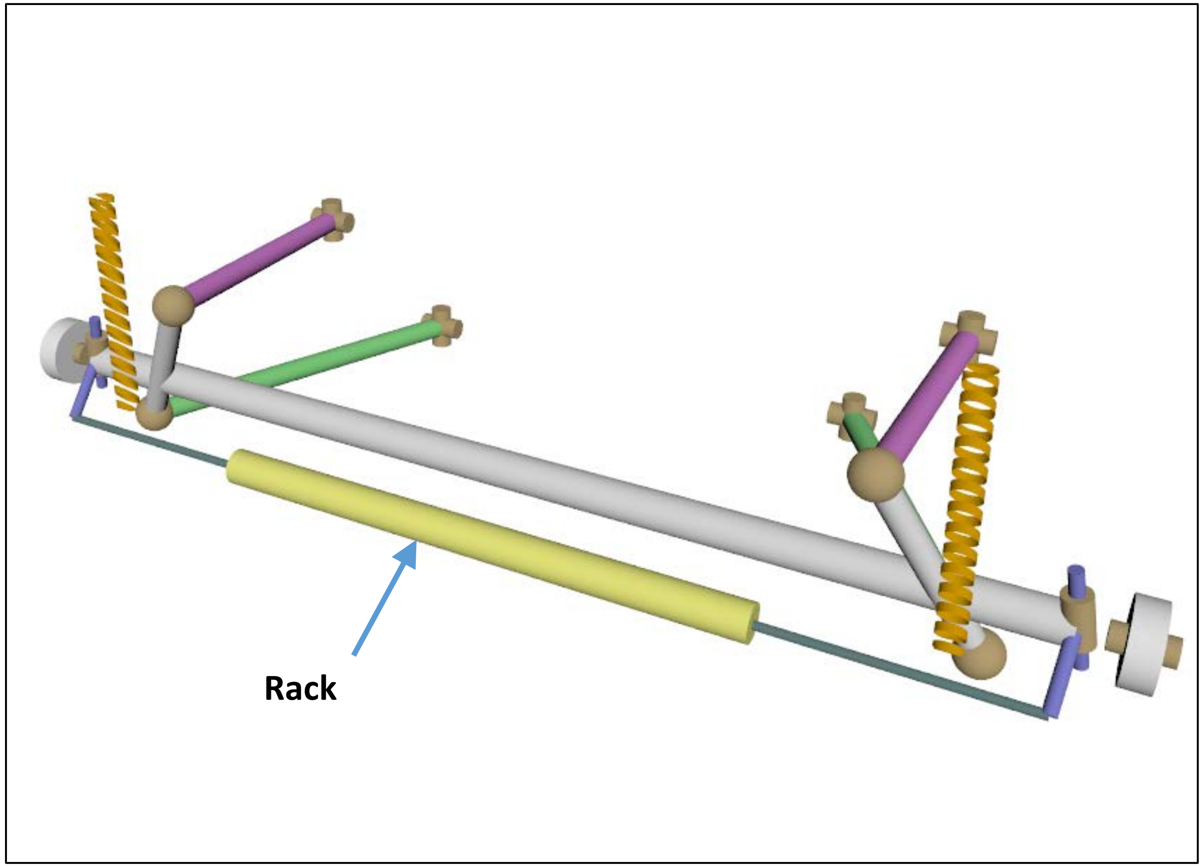
Direction of the axis of rotation

Maximum steering angle in degrees (float)

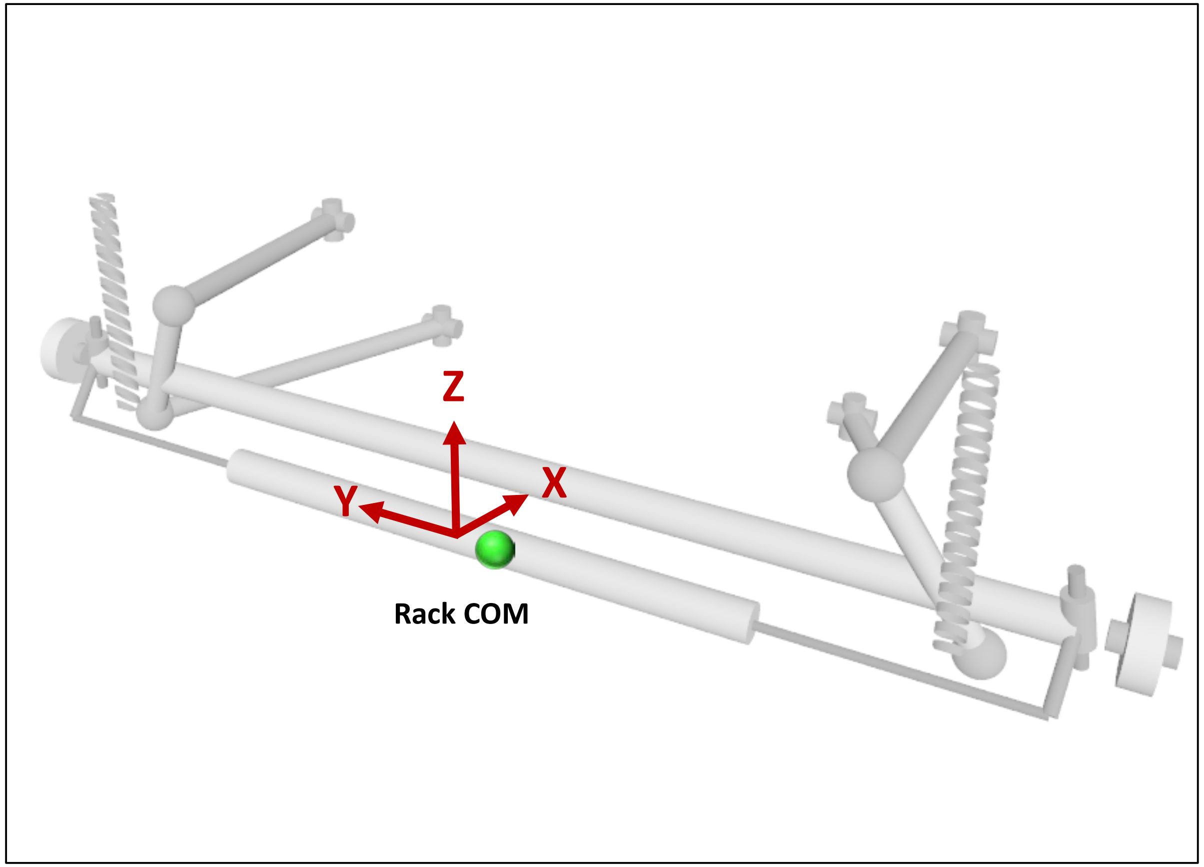
Steering Templates

Rack Pinion


RACK-PINION STEERING TEMPLATE
3D rigid body
joint
Distance constraint
Distance constraint
Steering Link
Steering input
Prismatic joint
Chassis



STEERING SUBSYSTEM
PARTS



STEERING SUBSYSTEM
HARDPOINTS

JSON specification file for rack-pinion

```
{
  "Name":                    “Name of this subsystem",
  "Type":                    "Steering",
  "Template":                "RackPinion",

  "Steering Link":
  {
    "Mass":                  1.0,
    "COM":                   1.0,
    "Inertia":               [1.0, 1.0, 1.0],
    "Radius":                1.0,
    "Length":                1.0
  },

  "Pinion":
  {
    "Radius":                1.0,
    "Maximum Angle":         1.0
  }
}
```

- "Name": Identifier name (string)
- "Type": Subsystem type (string)
- "Template": Template type (string)
- "Mass": Mass in kg (float)
- "COM": Lateral offset in m (float)
- "Inertia": Moments of inertia in kg m^2 (float)
- "Radius" (Steering Link): Visualization radius in m (float)
- "Length": Visualization length in m (float)
- "Radius" (Pinion): Pinion radius in m (float)
- "Maximum Angle": Maximum rotation angle in rad (float)

JSON specification file for rack-pinion

- User specifies all values, except those in red
- The reference frame for a rack-pinion steering subsystem is assumed to be aligned with the chassis reference frame and centered at the middle of the steering link
- The rack displacement is obtained as
$$d = r \cdot (\alpha_{max} \cdot s)$$
where r is the pinion radius, α_{max} is the maximum pinion angle, and s is the steering input ($s \in [-1,1]$)

Antiroll-bar Subsystem

ChAntirollBar base class

- Defines the common interface for any antiroll-bar subsystem
- All classes defining particular steering templates inherit from ChAntirollBar

```
///
/// Base class for an antiroll-bar subsystem.
///
class CH_VEHICLE_API ChAntirollBar : public ChPart
```

ChAntirollBar base class virtual methods

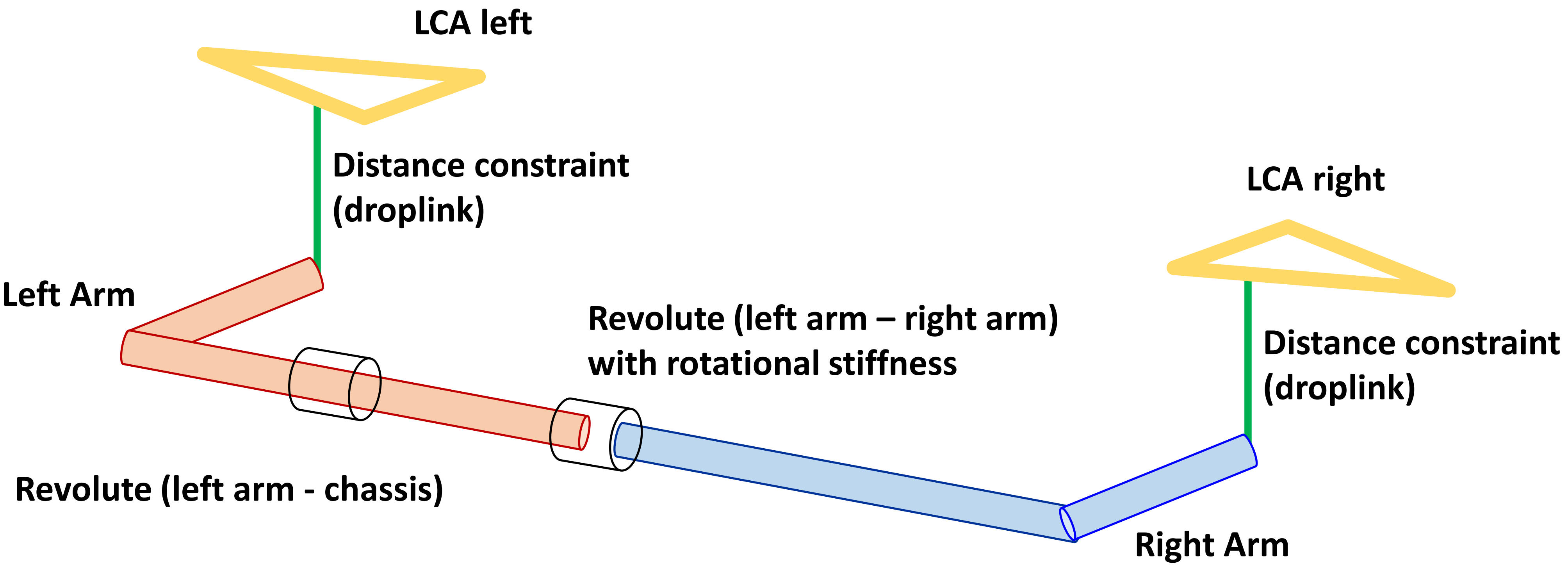
- Initialize the subsystem relative to the chassis body

```
    /// Initialize this anti-roll bar subsystem.
    /// The anti-roll bar subsystem is initialized by attaching it to the specified
    /// chassis body at the specified location (with respect to and expressed in
    /// the reference frame of the chassis). It is assumed that the suspension
    /// reference frame is always aligned with the chassis reference frame.
    /// Finally, susp_body_left and susp_body_right are handles to the suspension
    /// bodies to which the anti-roll bar's droplinks are to be attached.
    virtual void Initialize(
        std::shared_ptr<ChBodyAuxRef> chassis,   ///< [in] handle to the chassis body
        const ChVector<>& location,              ///< [in] location relative to the chassis frame
        std::shared_ptr<ChBody> susp_body_left,  ///< [in] susp body to which left droplink is connected
        std::shared_ptr<ChBody> susp_body_right  ///< [in] susp body to which right droplink is connected
        ) = 0;
```

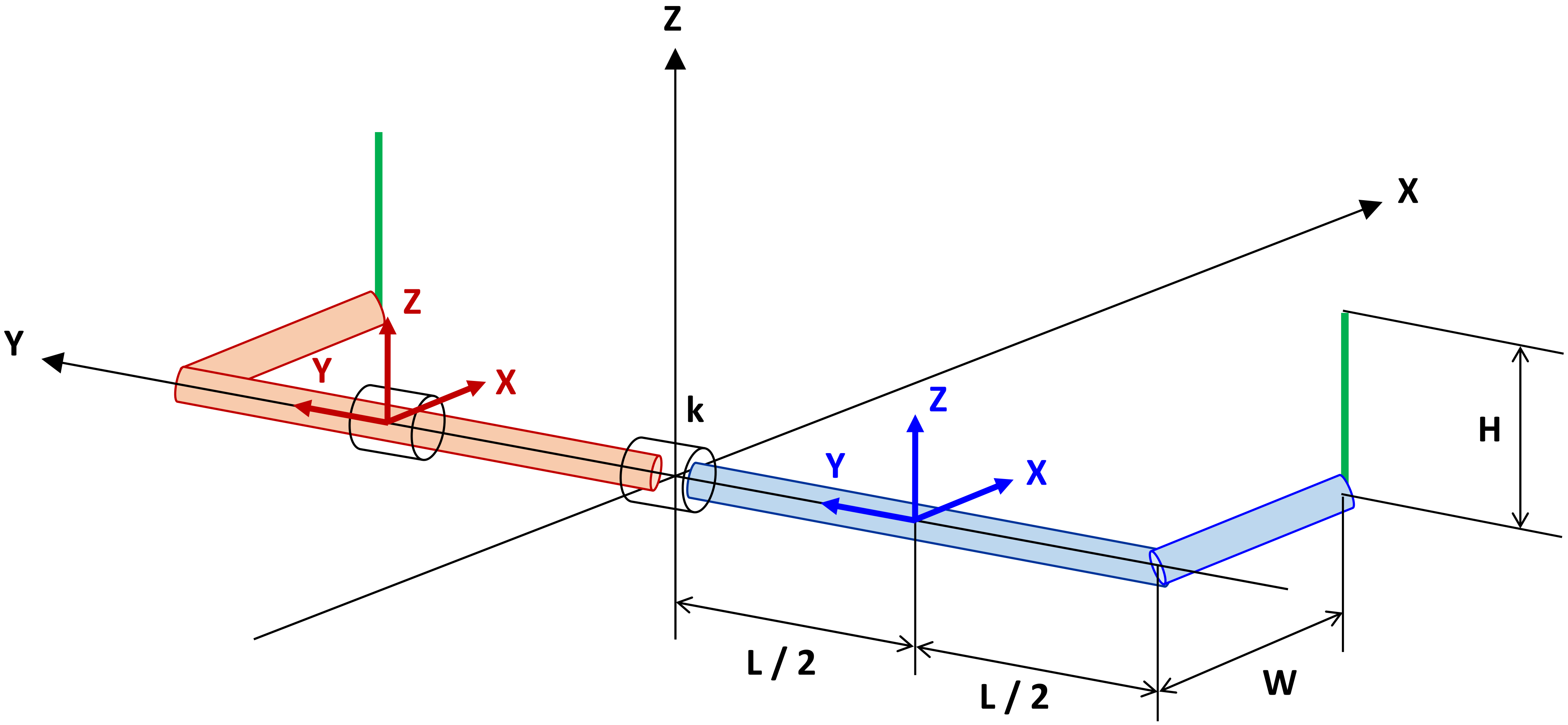
Antiroll-bar Templates

Antiroll-bar RSD

Simple anti-roll bar (model)



Simple anti-roll bar (template parameters)



Simple anti-roll bar (template parameters)

- Geometric
  - Arm length (L)
  - Arm width (W)
  - Arm radius (r) – visualization only
  - Droplink height (H)
- Mass properties
  - Arm mass (m)
  - Arm moments of inertia (Ixx)
- Stiffness
  - Spring coefficient (k)

- Subsystem reference frame (XYZ)
  - Assumed parallel to the chassis reference frame
- Left arm centroidal frame (XYZ) at $[0, L/2, 0]$
- Right arm centroidal frame (XYZ) at $[0, -L/2, 0]$

JSON specification file for RSD antiroll-bar

```
{
  "Name":                          "Generic RSD Antirollbar",
  "Type":                          "Antirollbar",
  "Template":                      "AntirollBarRSD",

  "Arm":
  {
    "Mass":                        1.0,
    "Inertia":                     [1, 1, 1],
    "Length":                      0.70,
    "Width":                       0.4,
    "Radius":                      0.02
  },

  "Droplink":
  {
    "Height":                      -0.04
  },

  "RSD":
  {
    "Spring Coefficient":           1000000,
    "Damping Coefficient":           2000
  }
}
```

Subsystem type (string)

Template type (string)

Spring and damper coefficients

Driveline Subsystem

ChDriveline base class

- Defines the common interface for any driveline subsystem
- All classes defining particular driveline templates inherit from ChDriveline

```
///
/// Base class for a driveline subsystem.
///
class CH_VEHICLE_API ChDriveline : public ChPart
```

ChDriveline base class members

- A ChDriveline has:

```
std::shared_ptr<ChShaft> m_driveshaft;  ///< handle to the shaft connection to the powertrain

std::vector<int> m_driven_axles;        ///< indexes of the driven vehicle axles
```

ChDriveline base class accessors and methods

- A ChDriveline provides access to:
  - Its constituent parts (driveshaft)
  - Angular speed of the driveshaft
  - Motor torque to be applied to a given vehicle wheel

- A ChDriveline provides a method to set the (input) torque from the powertrain system (typically invoked by the owning vehicle system)

```
/// Apply the specified motor torque.
/// This represents the input to the driveline subsystem from the powertrain
/// system.
void ApplyDriveshaftTorque(double torque)  { m_driveshaft->SetAppliedTorque(torque); }
```

ChDriveline base class virtual methods

- Initialize the driveline and connect it to the specified vehicle axles

```
/// Initialize the driveline subsystem.
/// This function connects this driveline subsystem to the axles of the
/// specified suspension subsystems.
virtual void Initialize(std::shared_ptr<ChBody> chassis,      ///< handle to the chassis body
                        const ChSuspensionList& suspensions,  ///< list of all vehicle suspension subsystems
                        const std::vector<int>& driven_axles  ///< indexes of the driven vehicle axles
                        ) = 0;
```

- Specify number of driven axles

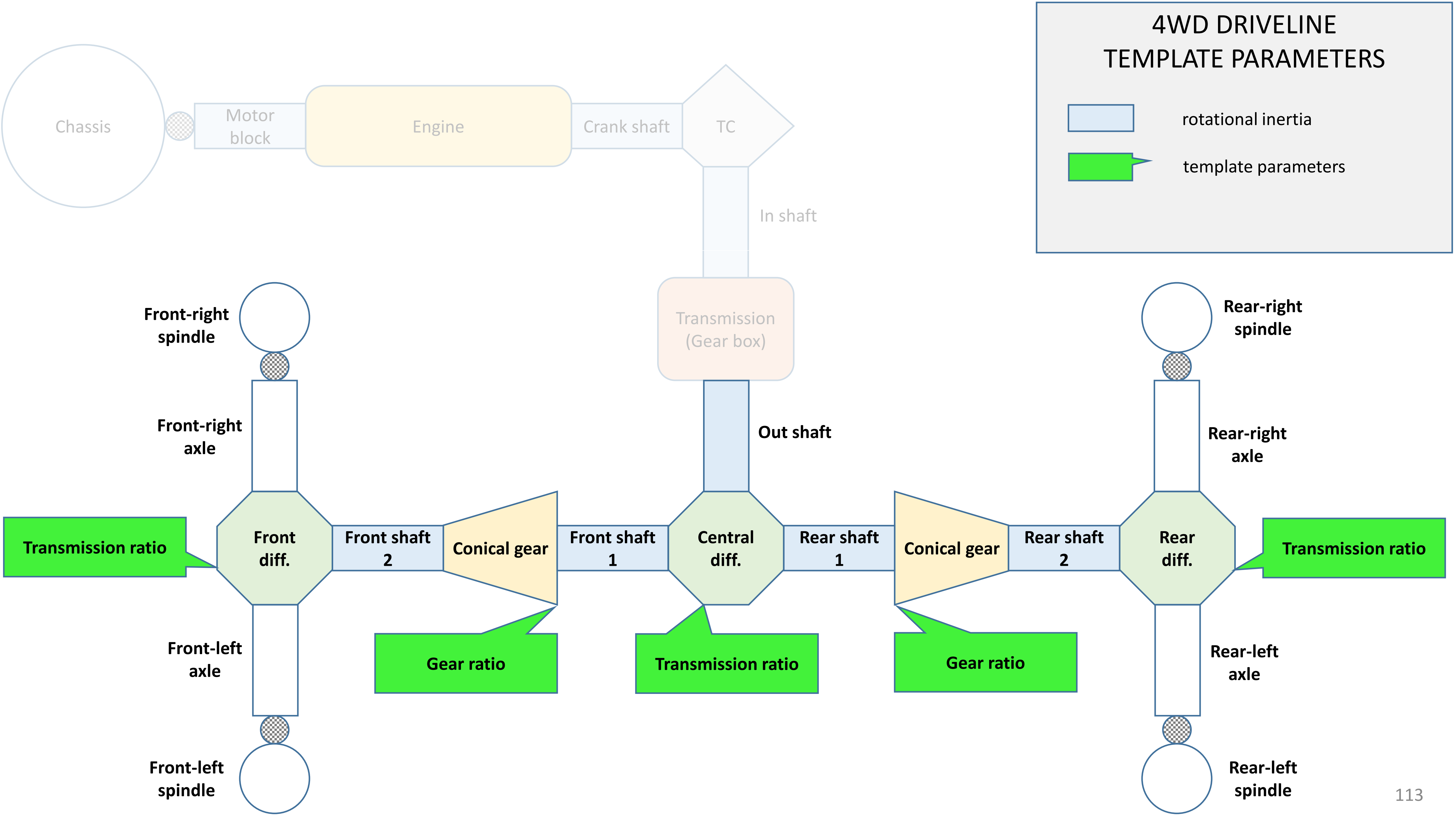
```
/// Return the number of driven axles.
virtual int GetNumDrivenAxles() const = 0;
```

- Set the (output) torque to be applied to the specified wheel

```
/// Get the motor torque to be applied to the specified wheel.
virtual double GetWheelTorque(const ChWheelID& wheel_id) const = 0;
```

Driveline Templates

ShaftsDriveline4WD


4WD DRIVELINE
TEMPLATE PARAMETERS
rotational inertia
template parameters
Chassis
Motor block
Engine
Crank shaft
TC
In shaft
Transmission (Gear box)
Out shaft
Front-right spindle
Front-right axle
Front diff.
Front shaft 2
Conical gear
Front shaft 1
Central diff.
Rear shaft 1
Conical gear
Rear shaft 2
Rear diff.
Rear-right spindle
Rear-right axle
Transmission ratio
Transmission ratio
Gear ratio
Transmission ratio
Gear ratio
Front-left axle
Front-left spindle
Rear-left axle
Rear-left spindle

JSON specification file for ShaftsDriveline4WD

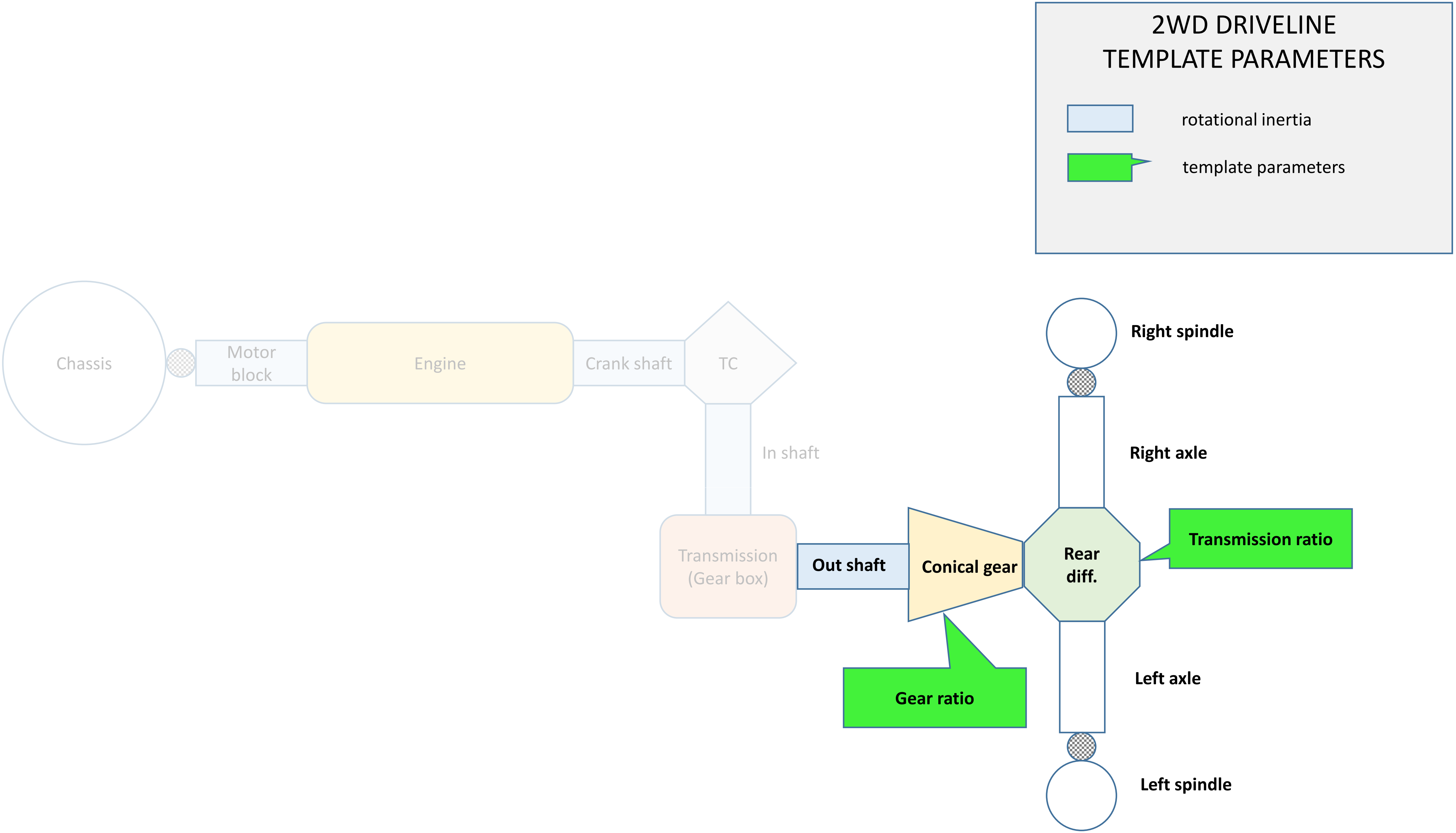
```
{
  "Name":                       "HMMWV AWD Driveline",
  "Type":                       "Driveline",
  "Template":                   "ShaftsDriveline4WD",

  "Shaft Direction":
  {
    "Motor Block":              [1, 0, 0],
    "Axle":                     [0, 1, 0]
  },

  "Shaft Inertia":
  {
    "Driveshaft":               0.5,
    "Front Driveshaft":         0.5,
    "Rear Driveshaft":          0.5,
    "Central Differential Box": 0.6,
    "Front Differential Box":   0.6,
    "Rear Differential Box":    0.6
  },

  "Gear Ratio":
  {
    "Front Conical Gear":       -0.2,
    "Rear Conical Gear":        -0.2,
    "Central Differential":     -1.0,
    "Front Differential":       -1.0,
    "Rear Differential":        -1.0
  }
}
```

Subsystem type (string)

Template type (string)

Driveline Templates

ShaftsDriveline2WD


2WD DRIVELINE
TEMPLATE PARAMETERS
rotational inertia
template parameters
Chassis
Motor block
Engine
Crank shaft
TC
In shaft
Transmission (Gear box)
Out shaft
Conical gear
Rear diff.
Transmission ratio
Gear ratio
Right spindle
Right axle
Left axle
Left spindle

JSON specification file for ShaftsDriveline2WD

```
{
  "Name":                         "HMMWV RWD Driveline",
  "Type":                         "Driveline",
  "Template":                     "ShaftsDriveline2WD",

  "Shaft Direction":
  {
    "Motor Block":                [1, 0, 0],
    "Axle":                       [0, 1, 0]
  },

  "Shaft Inertia":
  {
    "Driveshaft":                 0.5,
    "Differential Box":           0.6
  },

  "Gear Ratio":
  {
    "Conical Gear":               -0.2,
    "Differential":               -1.0
  }
}
```

Subsystem type (string)

Template type (string)

Sample simulations

ISO double lane change

Acceleration/braking test (Fiala tires)

Rigid mesh terrain