ROS Interface for Real-Time Vehicle Simulator

Junkil Park ([junkil.park@cis.upenn.edu](mailto:junkil.park@cis.upenn.edu))

Jialue Huang ([jialuehuang@gmail.com](mailto:jialuehuang@gmail.com))

Aleksey Nogin ([anogin@hrl.com](mailto:anogin@hrl.com))

Gavin Holland (gholland@hrl.com)

10/7/2013

# Overview

The figure below briefly illustrates how the real-time Vehicle Simulator and a controller will interact in ROS.

The table below is a list of sensors outputs provided by the Vehicle Simulator.

|  |  |  |
| --- | --- | --- |
| **Sensor** | **Topic** | **ROS Message type** |
| **GPS** | /abv/gps | sensor\_msgs/NavSatFix |
| **Wheel Encoder** | /abv/wheel\_encoder\_front  /abv/wheel\_econder\_rear | abv\_msgs/WheelEncoderFront  abv\_msgs/WheelEncoderRear |
| **Gear** | /abv/gear\_number  /abv/gear\_ratio | abv\_msgs/GearNumber  abv\_msgs/GearRatio |
| **IMU** | /abv/imu | abv\_msgs/IMU |
| **Speed (Phase 1 only)** | /sv/speed | abv\_msgs/Speed |
| *Additional Motion (Phase 2+)* | /abv/motion1 … /abv/motion5 | abv\_msgs/Motion |
| **Front Facing Radar**  *(UPenn - Phase 2+)*  **(Need in Phase 1 for CMU)** | /abv/radar | abv\_msgs/Radar |
| *Powertrain (Phase 2+)* | /abv/powertrain | abv\_msgs/Powertrain |

The table below is a list of message types for control input for the Vehicle Simulator.

|  |  |  |
| --- | --- | --- |
| **Control** | **Topic** | **ROS Message type** |
| **Acceleration** | /abv /control\_accel | abv\_msgs/ControlAccel |
| **Steering** | /abv /control\_steer | abv\_msgs/ControlSteer |

# ROS Message Type Definition

## std\_msgs/Header (included in each message type for time stamp)

|  |
| --- |
| # Standard metadata for higher-level stamped data types.  # This is generally used to communicate timestamped data  # in a particular coordinate frame.  #  # sequence ID: consecutively increasing ID  uint32 seq  #Two-integer timestamp that is expressed as:  # \* stamp.secs: seconds (stamp\_secs) since epoch  # \* stamp.nsecs: nanoseconds since stamp\_secs  # time-handling sugar is provided by the client library  time stamp  #Frame this data is associated with  # 0: no frame  # 1: global frame  string frame\_id |

In Phase 1 implementation:

* On VS to controller messages, seq and frame\_id are unset and stamp is set to ros::Time::now() at the time the ROS message is created by the CAN↔ROS gateway.
  + We may be able to use the time stamp from the CAN stack of the time the message was received by the black box CAN driver.
* On controller to VS messages, header is ignored by the CAN↔ROS gateway.

## sensor\_msgs/NavSatFix (for GPS data)

|  |
| --- |
| # Navigation Satellite fix for any Global Navigation Satellite System  #  # Specified using the WGS 84 reference ellipsoid  # header.stamp specifies the ROS time for this measurement (the  # corresponding satellite time may be reported using the  # sensor\_msgs/TimeReference message).  #  # header.frame\_id is the frame of reference reported by the satellite  # receiver, usually the location of the antenna. This is a  # Euclidean frame relative to the vehicle, not a reference  # ellipsoid.  Header header  # satellite fix status information  NavSatStatus status  # Latitude [degrees]. Positive is north of equator; negative is south.  float64 latitude  # Longitude [degrees]. Positive is east of prime meridian; negative is west.  float64 longitude  # Altitude [m]. Positive is above the WGS 84 ellipsoid  # (quiet NaN if no altitude is available).  float64 altitude  # Position covariance [m^2] defined relative to a tangential plane  # through the reported position. The components are East, North, and  # Up (ENU), in row-major order.  #  # Beware: this coordinate system exhibits singularities at the poles.  float64[9] position\_covariance  # If the covariance of the fix is known, fill it in completely. If the  # GPS receiver provides the variance of each measurement, put them  # along the diagonal. If only Dilution of Precision is available,  # estimate an approximate covariance from that.  uint8 COVARIANCE\_TYPE\_UNKNOWN = 0  uint8 COVARIANCE\_TYPE\_APPROXIMATED = 1  uint8 COVARIANCE\_TYPE\_DIAGONAL\_KNOWN = 2  uint8 COVARIANCE\_TYPE\_KNOWN = 3  uint8 position\_covariance\_type |

In Phase 1 code:

* float64 latitude – set from CarSim’s GPS\_Lat – “GPS Latitude offset of the vehicle sprung mass origin point relative to local reference” (i.e. ground truth of vehicle location)
* float64 longitude – set from CarSim’s GPS\_Lon – “GPS Longitude offset of the vehicle sprung mass origin point relative to local reference” (i.e. ground truth of vehicle location)
* Other fields unset
* Latitude and longitude truncated to 1ms (1/3.6E6 deg) precition, transmitted every 100ms

## abv\_msgs/WheelEncoderFront & abv\_msgs/WheelEncoderRear (for wheel encoder)

|  |
| --- |
| Header header  # CarSim’s Vx\_L1 (km/h)  # “Wheel L1 (equivalent) The equivalent wheel speed is defined as the spin  # (rad/sec) multiplied by the effective rolling radius. It is the translational  # speed that corresponds to the angular rate during free rolling.”  float32 front\_left\_encoder  # CarSim’s Vx\_R1 (km/h)  float32 front\_right\_encoder  Header header  # CarSim’s Vx\_L2 (km/h)  float32 rear\_left\_encoder  # CarSim’s Vx\_R2 (km/h)  float32 rear\_right\_encoder |

Note – the message format and semantics was chosen to be the closest match to the CAN messages available on the ABV.

Both messages truncated to precision of 1/32 km/h, transmitted every 50ms.

## abv\_msgs/GearNumber & abv\_msgs/GearRatio (for state variable of gear)

|  |
| --- |
| Header header  # CarSim’s GearStat:  # “Transmission gear based on internal model and imported variables.”  uint8 gear\_status  Header header  # CarSim’s Rgear\_Tr:  # “Transmission gear ratio”  float32 gear\_ratio |

Gear ratio truncated to precision of 1/32. Both messages transmitted every 25 ms.

## abv\_msgs/IMU (for vehicle acceleration data)

|  |
| --- |
| Header header  # CarSim’s AVz (deg/s):  # Vehicle yaw (body-fixed) –  # The [sz] component of the angular velocity of the sprung mass of the vehicle.  float32 yaw\_rate  # CarSim’s AAz (translated rad/s^2  deg/s^2):  # “Vehicle yaw (body-fixed) –  # The [sz] component of the angular acceleration of the sprung mass of the vehicle.”  float32 yaw\_accel  # CarSim’s Ax (translated g’s  m/s^2):  # “Longitudinal acceleration –  # The [x] component of the acceleration of the instant CG of the vehicle.”  float32 long\_accel  # CarSim’s Ay (translated g’s  m/s^2):  # “Lateral acceleration –  # The [y] component of the acceleration of the instant CG of the vehicle.”  floar32 lat\_accel |

Will be transmitted every 10ms (unless ROS code is unable to keep up with this rate)

* Yaw rate truncated to 0.024 deg/s increments
* Yaw acceleration truncated to 1/4 deg/s2 increments.
* Longitudinal/lateral acceleration truncated to 1/16 m/s2 increments

## abv\_msgs/Speed (for state variable of powertrain)

|  |
| --- |
| Header header  # CarSim’s Vx (km/h):  # “Longitudinal speed –  # the [x] component of the velocity of the instant CG of the vehicle”  float32 speed |

* In Phase 1 Code - transmitted every 100ms, truncated at 1/64 of km/h.
* This is a temporary Phase 1 message that will not be supported beyond Phase 1 (actual ABV does not have it – use wheel encoder instead).

## abv \_msgs/Radar ObjHeader & abv\_msgs/RadarObject(for radar data)

|  |
| --- |
| # Object data header message  Header header  unsigned char object\_count;  unsigned char message\_count;  unsigned char cycle\_duration;  # Object(1-N) data message  Header header  unsigned char id;  float range\_x;  float range\_y;  float velocity\_x;  float velocity\_y;  float acceleration\_x;  float size;  unsigned char status;  unsigned char relevant;  unsigned char lifetime; |

In Phase 1 Code:

* Will be derived from CarSim object sensors (whose outputs need to be added to UPenn/CMU CarSim test scenarios)

## abv\_msgs/Motion (for motion sensor data)

|  |
| --- |
| Header header  # TBD – which other components (besides those provided by IMU) will we need? |

Low priority – not planned to be used by TA 3 teams in in Phase 1.

## /abv \_msgs/Powertrain (for state variable of powertrain)

|  |
| --- |
| Header header  # CarSim’s AV\_Eng  # “Engine crankshaft” (rpm)  float32 engine\_speed |

Low priority – not planned to be used by TA 3 teams in in Phase 1.

## abv\_msgs/ControlAccel (for braking and throttle control input)

|  |
| --- |
| Header header  # CarSim’s IMP\_THROTTLE\_ENGINE: “Open loop throttle control”  float32 throttle  # CarSim’s IMP\_PCON\_BK: “Brake master cylinder pressure” (MPa)  float32 brake |

* In Phase 1:
  + Header will be ignored
  + Throttle is a value between 0 and 1 (the fraction of max throttle). Gateway will limit to range from 0 to 2, truncate at 1/218 and passed to CarSim’s IMP\_THROTTLE\_ENGINE
  + Brake is a value between 0 and 20, will be truncated at 0.01 and passed toI CarSim’s MP\_PCON\_BK inputs
* In Phase 2+, we plan to implement a custom brake and engine controller mimicking those in the ABV and use the same semantics for the message as in the real ACC:
  + Throttle is the value as an Axle Torque Request in Nm
  + Brake is the Braking Acceleration Request in m/s2

## abv\_msgs/ControlSteer (for steering control input)

|  |
| --- |
| Header header  # CarSim’s IMP\_STEER\_SW: “Steering wheel angle” (degrees)  float32 steering\_wheel\_angle |

* In Phase 1, header will be ignored
* steering\_wheel\_angle will be truncated at 1/16 degree increment and passed to CarSim’s IMP\_STEER\_SW