Motorcycle Awareness System (MAS)

ME553 Group EPE 6

Generated by Doxygen 1.8.6

Sun Apr 26 2015 19:01:54

Contents

1 Motorcycle Awareness System Overview

This low-level design documentation details a mockup of the Motorcycle Awareness System. It provides part of the functionality of the MAS for both the motorcycle and car. Mocking of input signals such as the radar signals, vehicle-to-vehicle (V2V) communication signals, and GPS signals mimics the interactions that the MAS would have with actual sensor data on the finished product.

The implemented functionality includes continuous tracking of the motorcycle using GPS, determining whether a hazard is present using the motorcycle's radar sensor signals, and relaying a warning to the motorcycle rider of a potential threat. For the car, the logic determines whether the car's blinker is on based on blinker signal data on the CAN bus, determines whether a motorcycle is in range and assess the potential danger, and issues a warning to the driver as necessary.

MAS Concept Sketch

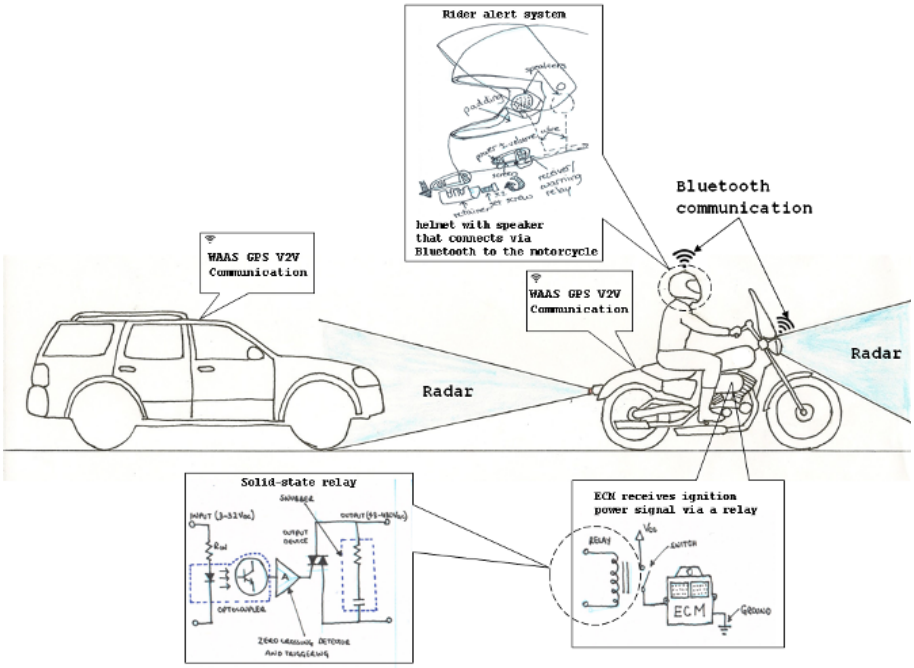


Figure 1: MAS concept sketch

2 Todo List

Member MotorcycleAwarenessSystem::GetMotorcycleLocation (void)

Acquire the data packet from the motorcycle

Member MotorcycleAwarenessSystem::IsMotorcycleInRange (void)

Process the data packet and determine threat

Member MotorcycleAwarenessSystem::RelayWarningToOperator (void)

Transmit the bluetooth message to the operator

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

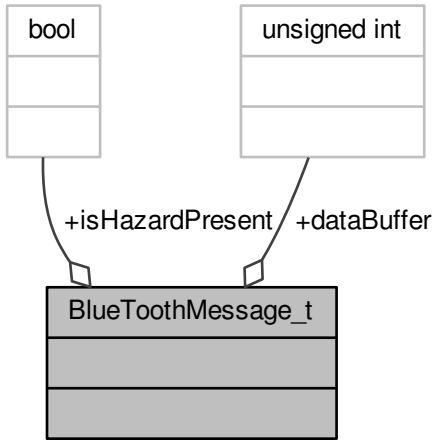
5 Class Documentation

5.1 BlueToothMessage_t Struct Reference

Struct for bluetooth message.

`#include <MotorcycleAwarenessSystemTypes.hpp>`

Collaboration diagram for BlueToothMessage_t:



Public Attributes

- bool isHazardPresent
  Hazard flag.
- unsigned int dataBuffer [255]
  Bluetooth data buffer.

5.1.1 Detailed Description

Struct for bluetooth message.

Definition at line 42 of file MotorcycleAwarenessSystemTypes.hpp.

5.1.2 Member Data Documentation

5.1.2.1 unsigned int BlueToothMessage_t::dataBuffer[255]

Bluetooth data buffer.

Definition at line 45 of file MotorcycleAwarenessSystemTypes.hpp.

5.1.2.2 bool BlueToothMessage_t::isHazardPresent

Hazard flag.

Definition at line 44 of file MotorcycleAwarenessSystemTypes.hpp.

The documentation for this struct was generated from the following file:

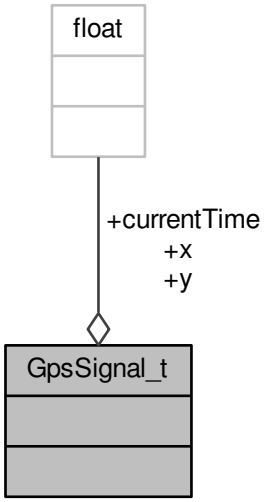
- MotorcycleAwarenessSystemTypes.hpp

5.2 GpsSignal_t Struct Reference

Structure that emulates a GPS signal.

`#include <MotorcycleAwarenessSystemTypes.hpp>`

Collaboration diagram for GpsSignal_t:



Public Attributes

- Coordinate_t x

  x-axis coordinate
- Coordinate_t y

  y-axis coordinate
- currentTime_t currentTime

  Current time at coordinates x,y.

5.2.1 Detailed Description

Structure that emulates a GPS signal.

Definition at line 25 of file MotorcycleAwarenessSystemTypes.hpp.

5.2.2 Member Data Documentation

5.2.2.1 currentTime_t GpsSignal_t::currentTime

Current time at coordinates x,y.

Definition at line 29 of file MotorcycleAwarenessSystemTypes.hpp.

5.2.2.2 Coordinate_t GpsSignal_t::x

x-axis coordinate

Definition at line 27 of file MotorcycleAwarenessSystemTypes.hpp.

5.2.2.3 Coordinate_t GpsSignal_t::y

y-axis coordinate

Definition at line 28 of file MotorcycleAwarenessSystemTypes.hpp.

The documentation for this struct was generated from the following file:

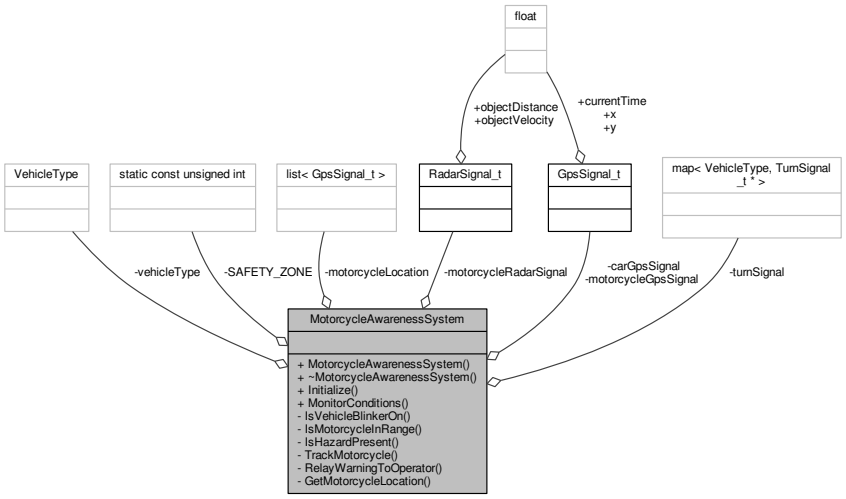
- MotorcycleAwarenessSystemTypes.hpp

5.3 MotorcycleAwarenessSystem Class Reference

Class declaration for the Motorcycle Awareness System (MAS)

`#include <MotorcycleAwarenessSystem.hpp>`

Collaboration diagram for MotorcycleAwarenessSystem:



Public Member Functions

- MotorcycleAwarenessSystem (VehicleType vehicleType)

  Constructor.

- ~MotorcycleAwarenessSystem (void)

  Destructor.
- bool Initialize (TurnSignal_t *motorcycleTurnSignal, TurnSignal_t *carTurnSignal, RadarSignal_t *motorcycleRadarSignal, GpsSignal_t *motorcycleGpsSignal, GpsSignal_t *carGpsSignal)
- void MonitorConditions (void)

  Method to continuously monitor the conditions during run-time.

Private Member Functions

- bool IsVehicleBlinkerOn (void)

  Method to determine whether the car's blinker is ON.
- bool IsMotorcycleInRange (void)

  Method to determine whether the motorcycle is within the car's range.
- bool IsHazardPresent (void)
- void TrackMotorcycle (void)

  Method to track the motorcycle using its GPS signal.
- void RelayWarningToOperator (void)

  Method to relay a warning to operator via bluetooth connectivity.
- MotorCycleLocation_t GetMotorcycleLocation (void)

Private Attributes

- std::list< GpsSignal_t > motorcycleLocation

  Container used to track the motorcycle's location.
- VehicleType vehicleType

  The vehicle type (motorcycle or car)
- std::map< VehicleType, TurnSignal_t * > turnSignal

  Storage for the turn signals.
- RadarSignal_t * motorcycleRadarSignal

  Pointer to a motorcycle radar signal.
- GpsSignal_t * motorcycleGpsSignal

  Pointer to a motorcycle GPS signal.
- GpsSignal_t * carGpsSignal

  Pointer to a car GPS signal.

Static Private Attributes

- static const unsigned int SAFETY_ZONE = 15U

5.3.1 Detailed Description

Class declaration for the Motorcycle Awareness System (MAS)

Definition at line 12 of file MotorcycleAwarenessSystem.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 MotorcycleAwarenessSystem::MotorcycleAwarenessSystem (VehicleType *vehicleType*)

Constructor.

Class definition for the Motorcycle Awareness System (MAS). This class processes various signals and interactions to realize the MAS

Definition at line 13 of file MotorcycleAwarenessSystem.cpp.

```
00014     :vehicleType( vehicleType )
00015 {
00016     // Do nothing
00017 }
```

5.3.2.2 MotorcycleAwarenessSystem::~MotorcycleAwarenessSystem (void)

Destructor.

Definition at line 20 of file MotorcycleAwarenessSystem.cpp.

```
00021 {
00022     // Do nothing
00023 }
```

5.3.3 Member Function Documentation

5.3.3.1 MotorCycleLocation_t MotorcycleAwarenessSystem::GetMotorcycleLocation (void) [private]

Todo Acquire the data packet from the motorcycle

Definition at line 163 of file MotorcycleAwarenessSystem.cpp.

```
00164 {
00165     // Dummy motorcycle location from V2V communication
00166     MotorCycleLocation_t motorCycleLocation;
00168
00169     return motorCycleLocation;
00170 }
```

Here is the caller graph for this function:

5.3.3.2 bool MotorcycleAwarenessSystem::Initialize (TurnSignal_t * *motorcycleTurnSignal*, TurnSignal_t * *carTurnSignal*, RadarSignal_t * *motorcycleRadarSignal*, GpsSignal_t * *motorcycleGpsSignal*, GpsSignal_t * *carGpsSignal*)

Method to initialize the MAS system

Parameters

in	*motorcycleTurnSignal*	Motorcycle turn signal
in	*carTurnSignal*	Car turn signal
in	*motorcycleRadarSignal*	Motorcycle radar signal
in	*motorcycleGpsSignal*	Motorcycle GPS signal
in	*carGpsSignal*	Car GPS signal

Returns

Pass fail status of the initialization

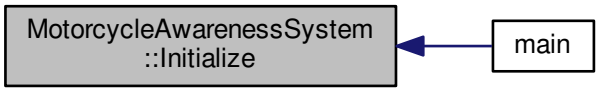
Return values

true	
false	

Definition at line 38 of file MotorcycleAwarenessSystem.cpp.

```
00041 {
00042     // Initialize the initialization state
00043     bool isSuccess = false;
00044 
00045     if ( motorcycleTurnSignal != (TurnSignal_t*)NULL &&
00046          carTurnSignal != (TurnSignal_t*)NULL &&
00047          motorcycleRadarSignal != (RadarSignal_t*)NULL &&
00048          motorcycleGpsSignal != (GpsSignal_t*)NULL &&
00049          carGpsSignal != (GpsSignal_t*)NULL )
00050     {
00051         // Initialize the motorcycle radar signal
00052         this->motorcycleRadarSignal = motorcycleRadarSignal;
00053 
00054         // Initialize the turn signal map
00055         turnSignal[MOTORCYCLE] = motorcycleTurnSignal;
00056         turnSignal[CAR] = carTurnSignal;
00057 
00058         // Initialize the GPS signals
00059         this->motorcycleGpsSignal = motorcycleGpsSignal;
00060         this->carGpsSignal = carGpsSignal;
00061 
00062         isSuccess = true;
00063     }
00064 
00065     return isSuccess;
00066 }
```

Here is the caller graph for this function:



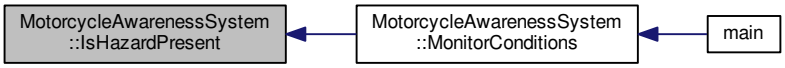
5.3.3.3 bool MotorcycleAwarenessSystem::IsHazardPresent (void) [private]

Method to determine whether a hazard is within the motorcycle's safety zone. An object within the safety zone is of potential danger to the motorcycle rider.

Definition at line 131 of file MotorcycleAwarenessSystem.cpp.

```
00132 {
00133     bool isHazardPresent = false;
00134
00135     if ( abs( this->motorcycleRadarSignal->objectDistance ) <=
      SAFETY_ZONE )
00136     {
00137         isHazardPresent = true;
00138     }
00139
00140     return isHazardPresent;
00141 }
```

Here is the caller graph for this function:



5.3.3.4 bool MotorcycleAwarenessSystem::IsMotorcycleInRange (void) [private]

Method to determine whether the motorcycle is within the car's range.

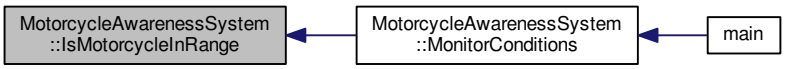
Todo Process the data packet and determine threat

Definition at line 109 of file MotorcycleAwarenessSystem.cpp.

```
00110 {
00111     bool isInRange = false;
00112
00113     // Determine where the motorcycle is relative to the car using the GPS signals
00114     if ( abs( (this->carGpsSignal->x) - (this->motorcycleGpsSignal->x) ) <=
      SAFETY_ZONE &&
00115          abs( (this->carGpsSignal->y) - (this->motorcycleGpsSignal->y) ) <=
      SAFETY_ZONE )
00116     {
00117         isInRange = true;
00118     }
00119     else
00120     {
00121         // Analyze the V2V data for a threat
00122         MotorCycleLocation_t motorCycleLocation =
      GetMotorcycleLocation();
00124     }
00125
00126     return isInRange;
00127 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



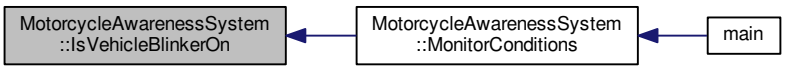
5.3.3.5 bool MotorcycleAwarenessSystem::IsVehicleBlinkerOn (void) [private]

Method to determine whether the car's blinker is ON.

Definition at line 95 of file MotorcycleAwarenessSystem.cpp.

```
00096 {
00097     bool isBlinkerOn = false;
00098
00099     if ( this->turnSignal[vehicleType]->isRightBlinkerOn ||
00100          this->turnSignal[vehicleType]->isLeftBlinkerOn )
00101     {
00102         isBlinkerOn = true;
00103     }
00104
00105     return isBlinkerOn;
00106 }
```

Here is the caller graph for this function:



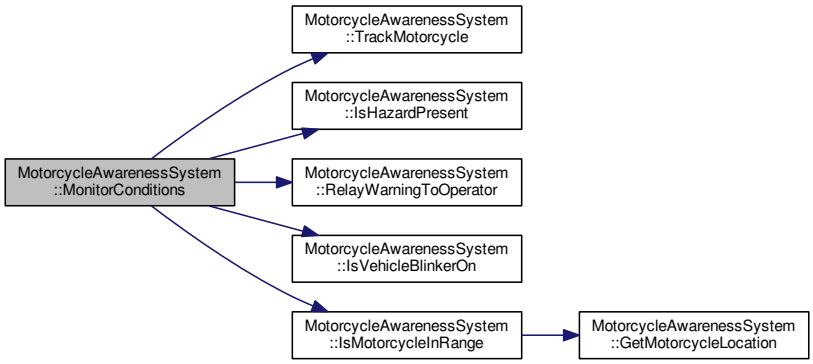
5.3.3.6 void MotorcycleAwarenessSystem::MonitorConditions (void)

Method to continuously monitor the conditions during run-time.

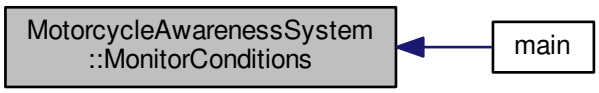
Definition at line 69 of file MotorcycleAwarenessSystem.cpp.

```
00070 {
00071     if ( MOTORCYCLE == vehicleType )
00072     {
00073         // Track the motorcycle
00074         TrackMotorcycle();
00075         // Check for hazards
00076         if ( IsHazardPresent() == true )
00077         {
00078             // Warn the motorcycle operator
00079             RelayWarningToOperator();
00080         }
00081     }
00082     // vehicleType == CAR
00083     else
00084     {
00085         // Check for hazards
00086         if ( (IsVehicleBlinkerOn() == true) && (
      IsMotorcycleInRange() == true) )
00087         {
00088             // Relay message to the car driver
00089             RelayWarningToOperator();
00090         }
00091     }
00092 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.7 void MotorcycleAwarenessSystem::RelayWarningToOperator (void) [private]

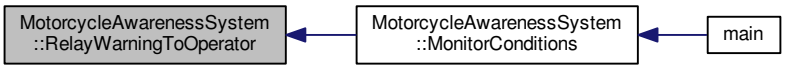
Method to relay a warning to operator via bluetooth connectivity.

Todo Transmit the bluetooth message to the operator

Definition at line 151 of file MotorcycleAwarenessSystem.cpp.

```
00152 {
00153     // Assemble the message to be sent
00154     BlueToothMessage_t blueToothMessage;
00155     blueToothMessage.isHazardPresent = true;
00156     // Dummy message
00157     blueToothMessage.dataBuffer[0] = 0x2015;
00158
00160 }
```

Here is the caller graph for this function:



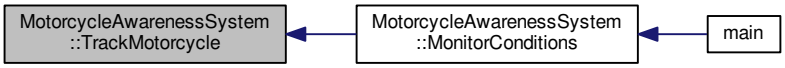
5.3.3.8 void MotorcycleAwarenessSystem::TrackMotorcycle (void) [private]

Method to track the motorcycle using its GPS signal.

Definition at line 144 of file MotorcycleAwarenessSystem.cpp.

```
00145 {
00146     // Push the motorcycle's GPS location onto the list
00147     motorcycleLocation.push_front( *motorcycleGpsSignal );
00148 }
```

Here is the caller graph for this function:



5.3.4 Member Data Documentation

5.3.4.1 GpsSignal_t* MotorcycleAwarenessSystem::carGpsSignal [private]

Pointer to a car GPS signal.

Definition at line 30 of file MotorcycleAwarenessSystem.hpp.

5.3.4.2 GpsSignal_t* MotorcycleAwarenessSystem::motorcycleGpsSignal [private]

Pointer to a motorcycle GPS signal.

Definition at line 29 of file MotorcycleAwarenessSystem.hpp.

5.3.4.3 std::list<GpsSignal_t> MotorcycleAwarenessSystem::motorcycleLocation [private]

Container used to track the motorcycle's location.

Definition at line 23 of file MotorcycleAwarenessSystem.hpp.

5.3.4.4 RadarSignal_t* MotorcycleAwarenessSystem::motorcycleRadarSignal [private]

Pointer to a motorcycle radar signal.

Definition at line 28 of file MotorcycleAwarenessSystem.hpp.

5.3.4.5 const unsigned int MotorcycleAwarenessSystem::SAFETY_ZONE = 15U [static], [private]

Distance from the radar sensor in feet in which a detected object becomes a potential danger

Definition at line 24 of file MotorcycleAwarenessSystem.hpp.

5.3.4.6 std::map<VehicleType, TurnSignal_t*> MotorcycleAwarenessSystem::turnSignal [private]

Storage for the turn signals.

Definition at line 27 of file MotorcycleAwarenessSystem.hpp.

5.3.4.7 VehicleType MotorcycleAwarenessSystem::vehicleType [private]

The vehicle type (motorcycle or car)

Definition at line 26 of file MotorcycleAwarenessSystem.hpp.

The documentation for this class was generated from the following files:

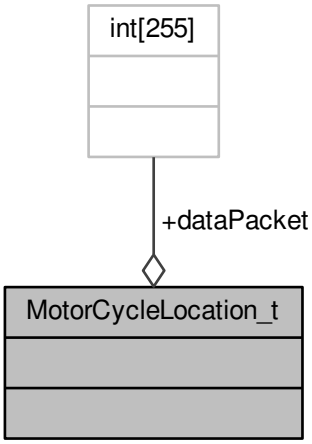
- MotorcycleAwarenessSystem.hpp
- MotorcycleAwarenessSystem.cpp

5.4 MotorCycleLocation_t Struct Reference

Struct for the V2V data.

`#include <MotorcycleAwarenessSystemTypes.hpp>`

Collaboration diagram for MotorCycleLocation_t:



Public Attributes

- DataPacket_t dataPacket

  V2V communication data packet.

5.4.1 Detailed Description

Struct for the V2V data.

Definition at line 50 of file MotorcycleAwarenessSystemTypes.hpp.

5.4.2 Member Data Documentation

5.4.2.1 DataPacket_t MotorCycleLocation_t::dataPacket

V2V communication data packet.

Definition at line 52 of file MotorcycleAwarenessSystemTypes.hpp.

The documentation for this struct was generated from the following file:

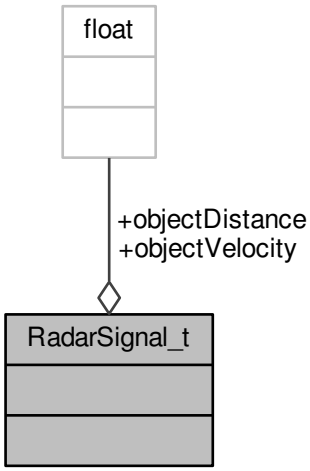
- MotorcycleAwarenessSystemTypes.hpp

5.5 RadarSignal_t Struct Reference

Structure that emulates a Radar signal.

```
#include <MotorcycleAwarenessSystemTypes.hpp>
```

Collaboration diagram for RadarSignal_t:



Public Attributes

- ObjectDistance objectDistance

  Distance from sensed object to radar sensor.
- ObjectVelocity objectVelocity

  Velocity of sensed object.

5.5.1 Detailed Description

Structure that emulates a Radar signal.

Definition at line 35 of file MotorcycleAwarenessSystemTypes.hpp.

5.5.2 Member Data Documentation

5.5.2.1 ObjectDistance RadarSignal_t::objectDistance

Distance from sensed object to radar sensor.

Definition at line 37 of file MotorcycleAwarenessSystemTypes.hpp.

5.5.2.2 ObjectVelocity RadarSignal_t::objectVelocity

Velocity of sensed object.

Definition at line 38 of file MotorcycleAwarenessSystemTypes.hpp.

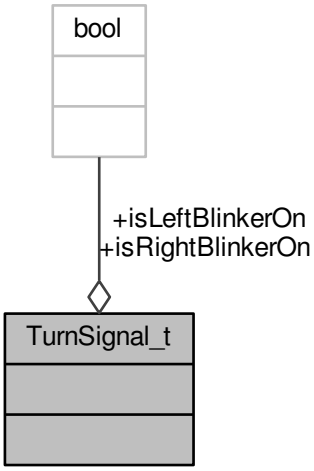
The documentation for this struct was generated from the following file:

- MotorcycleAwarenessSystemTypes.hpp

5.6 TurnSignal_t Struct Reference

`#include <MotorcycleAwarenessSystemTypes.hpp>`

Collaboration diagram for TurnSignal_t:



Public Attributes

- bool isRightBlinkerOn

  Right-hand-side blinker signal.
- bool isLeftBlinkerOn

  Left-hand-side blinker signal.

5.6.1 Detailed Description

Structure used to store the data coming from the turn-signal relay indicating the status of the blinker lights

Definition at line 16 of file MotorcycleAwarenessSystemTypes.hpp.

5.6.2 Member Data Documentation

5.6.2.1 bool TurnSignal_t::isLeftBlinkerOn

Left-hand-side blinker signal.

Definition at line 19 of file MotorcycleAwarenessSystemTypes.hpp.

5.6.2.2 bool TurnSignal_t::isRightBlinkerOn

Right-hand-side blinker signal.

Definition at line 18 of file MotorcycleAwarenessSystemTypes.hpp.

The documentation for this struct was generated from the following file:

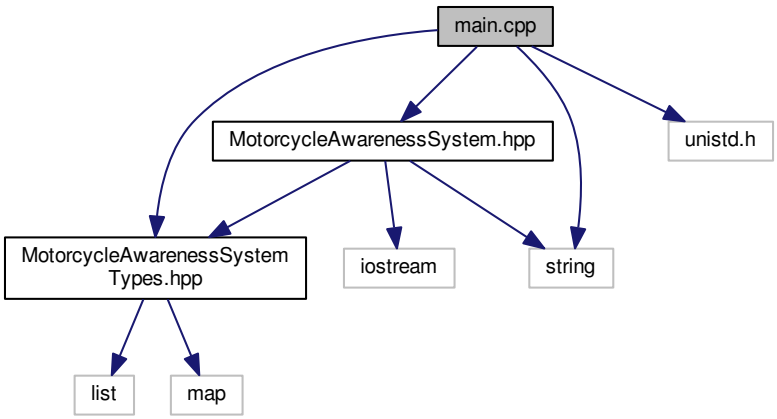
- MotorcycleAwarenessSystemTypes.hpp

6 File Documentation

6.1 main.cpp File Reference

```
#include "MotorcycleAwarenessSystem.hpp"
#include "MotorcycleAwarenessSystemTypes.hpp"
#include <string>
#include <unistd.h>
```

Include dependency graph for main.cpp:



Functions

- int main (void)

6.1.1 Function Documentation

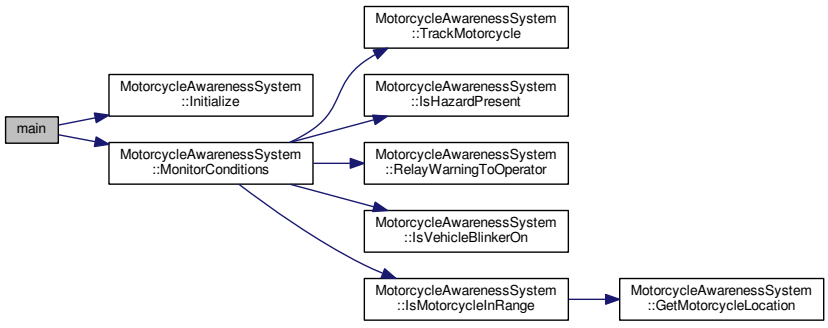
6.1.1.1 int main (void)

Definition at line 10 of file main.cpp.

```
00011 {
00012     // Define mock signals
00013     TurnSignal_t* motorcycleTurnSignal = new TurnSignal_t;
```

```
00014     TurnSignal_t* carTurnSignal = new TurnSignal_t;
00015     RadarSignal_t* motorcycleRadarSignal = new RadarSignal_t;
00016     GpsSignal_t* motorcycleGpsSignal = new GpsSignal_t;
00017     GpsSignal_t* carGpsSignal = new GpsSignal_t;
00018
00019     bool isSuccess = false;
00020
00021     // Instantiate the Motorcycle Awareness System (MAS) for the motorcycle
00022     MotorcycleAwarenessSystem MAS_motorcycle(
      MOTORCYCLE );
00023     // Initialize the car MAS
00024     isSuccess = MAS_motorcycle.Initialize( motorcycleTurnSignal, (TurnSignal_t*)NULL,
      motorcycleRadarSignal,
00025                                            motorcycleGpsSignal, carGpsSignal );
00026 #ifdef MAS_DEBUG
00027     if ( !isSuccess )
00028     {
00029         std::cout << "MAS initialization for the motorcycle failed" << std::endl;
00030     }
00031 #endif
00032
00033     // Instantiate the Motorcycle Awareness System (MAS) for the car
00034     MotorcycleAwarenessSystem MAS_car( CAR );
00035     // Initialize the car MAS
00036     isSuccess =  MAS_car.Initialize( (TurnSignal_t*)NULL, carTurnSignal, motorcycleRadarSignal,
00037                                      motorcycleGpsSignal, carGpsSignal );
00038 #ifdef MAS_DEBUG
00039     if ( !isSuccess )
00040     {
00041         std::cout << "MAS initialization for the car failed" << std::endl;
00042     }
00043 #endif
00044
00045     // Run the MAS systems for the car & motorcycle only if the systems were
00046     // initialized successfully
00047     if ( isSuccess )
00048     {
00049         while ( true )
00050         {
00051             MAS_motorcycle.MonitorConditions();
00052             MAS_car.MonitorConditions();
00053             usleep(250);
00054
00055 #ifdef MAS_DEBUG
00056             std::cout << "The MAS is running" << std::endl;
00057 #endif
00058         }
00059     }
00060
00061     return 0;
00062 }
```

Here is the call graph for this function:



6.2 main.cpp

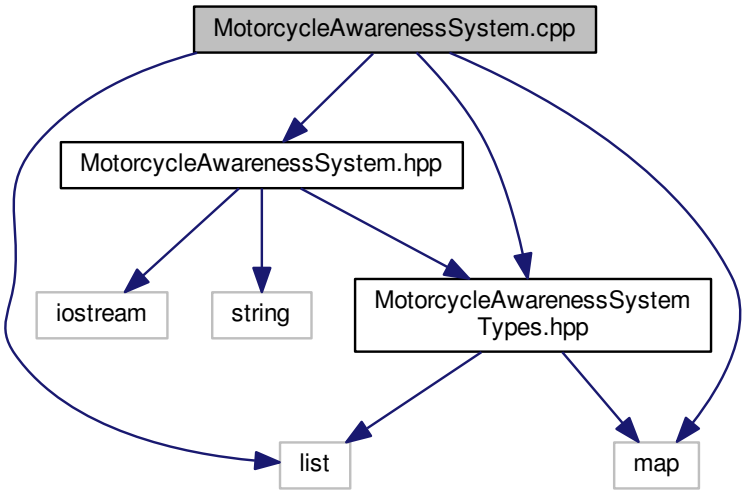
```
00001 #include "MotorcycleAwarenessSystem.hpp"
00002 #include "MotorcycleAwarenessSystemTypes.hpp"
00003
00004 #ifdef MAS_DEBUG
00005 #include <iostream>
00006 #endif
00007 #include <string>
00008 #include <unistd.h>
00009
00010 int main( void )
00011 {
00012     // Define mock signals
00013     TurnSignal_t* motorcycleTurnSignal = new TurnSignal_t;
00014     TurnSignal_t* carTurnSignal = new TurnSignal_t;
00015     RadarSignal_t* motorcycleRadarSignal = new RadarSignal_t;
00016     GpsSignal_t* motorcycleGpsSignal = new GpsSignal_t;
00017     GpsSignal_t* carGpsSignal = new GpsSignal_t;
00018
00019     bool isSuccess = false;
00020
00021     // Instantiate the Motorcycle Awareness System (MAS) for the motorcycle
00022     MotorcycleAwarenessSystem MAS_motorcycle(
      MOTORCYCLE );
00023     // Initialize the car MAS
00024     isSuccess = MAS_motorcycle.Initialize( motorcycleTurnSignal, (
      TurnSignal_t*)NULL, motorcycleRadarSignal,
00025                                            motorcycleGpsSignal, carGpsSignal );
00026 #ifdef MAS_DEBUG
00027     if ( !isSuccess )
00028     {
00029         std::cout << "MAS initialization for the motorcycle failed" << std::endl;
00030     }
00031 #endif
00032
00033     // Instantiate the Motorcycle Awareness System (MAS) for the car
00034     MotorcycleAwarenessSystem MAS_car( CAR );
00035     // Initialize the car MAS
00036     isSuccess =  MAS_car.Initialize( (TurnSignal_t*)NULL, carTurnSignal,
      motorcycleRadarSignal,
00037                                      motorcycleGpsSignal, carGpsSignal );
00038 #ifdef MAS_DEBUG
00039     if ( !isSuccess )
00040     {
00041         std::cout << "MAS initialization for the car failed" << std::endl;
00042     }
00043 #endif
00044
00045     // Run the MAS systems for the car & motorcycle only if the systems were
00046     // initialized successfully
00047     if ( isSuccess )
00048     {
00049         while ( true )
00050         {
00051             MAS_motorcycle.MonitorConditions();
00052             MAS_car.MonitorConditions();
00053             usleep(250);
00054
00055 #ifdef MAS_DEBUG
00056             std::cout << "The MAS is running" << std::endl;
00057 #endif
00058         }
00059     }
00060
00061     return 0;
00062 }
```

6.3 mainpage.dox File Reference

6.4 MotorcycleAwarenessSystem.cpp File Reference

```
#include "MotorcycleAwarenessSystem.hpp"
#include "MotorcycleAwarenessSystemTypes.hpp"
#include <list>
#include <map>
```

Include dependency graph for MotorcycleAwarenessSystem.cpp:



6.5 MotorcycleAwarenessSystem.cpp

```
00001 #include "MotorcycleAwarenessSystem.hpp"
00007 #include "MotorcycleAwarenessSystemTypes.hpp"
00008
00009 #include <list>
00010 #include <map>
00011
00013 MotorcycleAwarenessSystem::MotorcycleAwarenessSystem(
      VehicleType vehicleType )
00014     :vehicleType( vehicleType )
00015 {
00016     // Do nothing
00017 }
00018
00020 MotorcycleAwarenessSystem::~MotorcycleAwarenessSystem(
       void )
00021 {
00022     // Do nothing
00023 }
00024
00037
00038 bool MotorcycleAwarenessSystem::Initialize(
      TurnSignal_t* motorcycleTurnSignal, TurnSignal_t* carTurnSignal,
00039                                         RadarSignal_t* motorcycleRadarSignal,
      GpsSignal_t* motorcycleGpsSignal,
00040                                         GpsSignal_t* carGpsSignal  )
00041 {
00042     // Initialize the initialization state
00043     bool isSuccess = false;
00044
00045     if ( motorcycleTurnSignal != (TurnSignal_t*)NULL &&
```

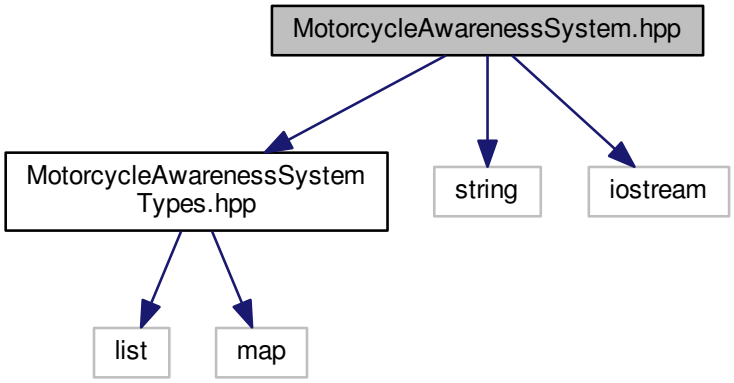
```
00046          carTurnSignal != (TurnSignal_t*)NULL &&
00047          motorcycleRadarSignal != (RadarSignal_t*)NULL &&
00048          motorcycleGpsSignal != (GpsSignal_t*)NULL &&
00049          carGpsSignal != (GpsSignal_t*)NULL )
00050     {
00051         // Initialize the motorcycle radar signal
00052         this->motorcycleRadarSignal = motorcycleRadarSignal;
00053 
00054         // Initialize the turn signal map
00055         turnSignal[MOTORCYCLE] = motorcycleTurnSignal;
00056         turnSignal[CAR] = carTurnSignal;
00057 
00058         // Initialize the GPS signals
00059         this->motorcycleGpsSignal = motorcycleGpsSignal;
00060         this->carGpsSignal = carGpsSignal;
00061 
00062         isSuccess = true;
00063     }
00064 
00065     return isSuccess;
00066 }
00067 
00069 void MotorcycleAwarenessSystem::MonitorConditions( void )
00070 {
00071     if ( MOTORCYCLE == vehicleType )
00072     {
00073         // Track the motorcycle
00074         TrackMotorcycle();
00075         // Check for hazards
00076         if ( IsHazardPresent() == true )
00077         {
00078             // Warn the motorcycle operator
00079             RelayWarningToOperator();
00080         }
00081     }
00082     // vehicleType == CAR
00083     else
00084     {
00085         // Check for hazards
00086         if ( (IsVehicleBlinkerOn() == true) && (
      IsMotorcycleInRange() == true) )
00087         {
00088             // Relay message to the car driver
00089             RelayWarningToOperator();
00090         }
00091     }
00092 }
00093 
00095 bool MotorcycleAwarenessSystem::IsVehicleBlinkerOn( void )
00096 {
00097     bool isBlinkerOn = false;
00098 
00099     if ( this->turnSignal[vehicleType]->isRightBlinkerOn ||
00100          this->turnSignal[vehicleType]->isLeftBlinkerOn )
00101     {
00102         isBlinkerOn = true;
00103     }
00104 
00105     return isBlinkerOn;
00106 }
00107 
00109 bool MotorcycleAwarenessSystem::IsMotorcycleInRange( void )
00110 {
00111     bool isInRange = false;
00112 
00113     // Determine where the motorcycle is relative to the car using the GPS signals
00114     if ( abs( (this->carGpsSignal->x) - (this->motorcycleGpsSignal->x) ) <=
      SAFETY_ZONE &&
00115          abs( (this->carGpsSignal->y) - (this->motorcycleGpsSignal->y) ) <=
      SAFETY_ZONE )
00116     {
00117         isInRange = true;
00118     }
00119     else
00120     {
00121         // Analyze the V2V data for a threat
00122         MotorCycleLocation_t motorCycleLocation =
      GetMotorcycleLocation();
00124     }
00125 
00126     return isInRange;
```

```
00127 }
00128
00131 bool MotorcycleAwarenessSystem::IsHazardPresent( void )
00132 {
00133     bool isHazardPresent = false;
00134
00135     if ( abs( this->motorcycleRadarSignal->objectDistance ) <=
      SAFETY_ZONE )
00136     {
00137         isHazardPresent = true;
00138     }
00139
00140     return isHazardPresent;
00141 }
00142
00144 void MotorcycleAwarenessSystem::TrackMotorcycle( void )
00145 {
00146     // Push the motorcycle's GPS location onto the list
00147     motorcycleLocation.push_front( *motorcycleGpsSignal );
00148 }
00149
00151 void MotorcycleAwarenessSystem::RelayWarningToOperator(
      void )
00152 {
00153     // Assemble the message to be sent
00154     BlueToothMessage_t blueToothMessage;
00155     blueToothMessage.isHazardPresent = true;
00156     // Dummy message
00157     blueToothMessage.dataBuffer[0] = 0x2015;
00158
00160 }
00161
00162 // Method to Get motorcycle's location via V2V communication
00163 MotorCycleLocation_t
      MotorcycleAwarenessSystem::GetMotorcycleLocation( void )
00164 {
00165     // Dummy motorcycle location from V2V communication
00166     MotorCycleLocation_t motorCycleLocation;
00168
00169     return motorCycleLocation;
00170 }
```

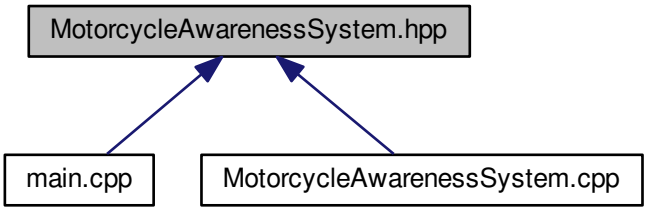
6.6 MotorcycleAwarenessSystem.hpp File Reference

```
#include "MotorcycleAwarenessSystemTypes.hpp"
#include <string>
#include <iostream>
```

Include dependency graph for MotorcycleAwarenessSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class MotorcycleAwarenessSystem

  Class declaration for the Motorcycle Awareness System (MAS)

6.7 MotorcycleAwarenessSystem.hpp

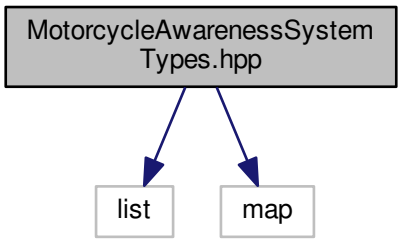
```
00001 #ifndef MOTORCYCLEAWARENESSSYSTEM_HPP
00005 #define MOTORCYCLEAWARENESSSYSTEM_HPP
00006
00007 #include "MotorcycleAwarenessSystemTypes.hpp"
00008
00009 #include <string>
00010 #include <iostream>
00011
```

```
00012 class MotorcycleAwarenessSystem
00013 {
00014     public:
00015         MotorcycleAwarenessSystem( VehicleType
      vehicleType );
00016         ~MotorcycleAwarenessSystem( void );
00017         bool Initialize( TurnSignal_t* motorcycleTurnSignal,
      TurnSignal_t* carTurnSignal,
00018                          RadarSignal_t* motorcycleRadarSignal,
      GpsSignal_t* motorcycleGpsSignal,
00019                          GpsSignal_t* carGpsSignal );
00020         void MonitorConditions( void );
00021
00022     private:
00023         std::list<GpsSignal_t> motorcycleLocation;
00024         static const unsigned int SAFETY_ZONE = 15U;
00025         VehicleType vehicleType;
00027         std::map<VehicleType, TurnSignal_t*> turnSignal;
00028         RadarSignal_t* motorcycleRadarSignal;
00029         GpsSignal_t* motorcycleGpsSignal;
00030         GpsSignal_t* carGpsSignal;
00031
00032         bool IsVehicleBlinkerOn( void );
00033         bool IsMotorcycleInRange( void );
00034         bool IsHazardPresent( void );
00035         void TrackMotorcycle( void );
00036         void RelayWarningToOperator( void );
00037         MotorCycleLocation_t GetMotorcycleLocation( void );
00038 };
00039 #endif
```

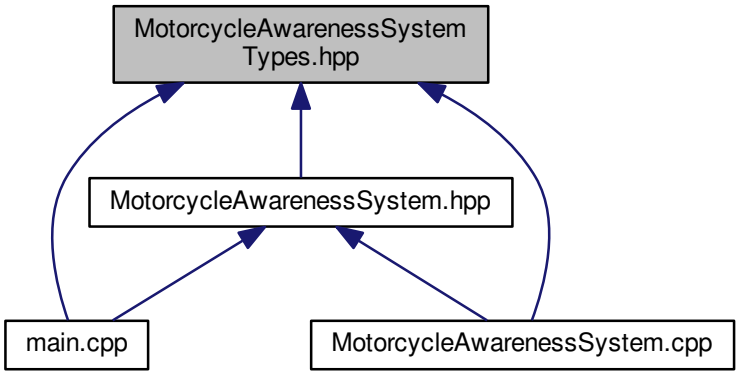
6.8 MotorcycleAwarenessSystemTypes.hpp File Reference

```
#include <list>
#include <map>
```

Include dependency graph for MotorcycleAwarenessSystemTypes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct TurnSignal_t
- struct GpsSignal_t

  Structure that emulates a GPS signal.
- struct RadarSignal_t

  Structure that emulates a Radar signal.
- struct BlueToothMessage_t

  Struct for bluetooth message.
- struct MotorCycleLocation_t

  Struct for the V2V data.

Typedefs

- typedef float Coordinate_t

  GPS coordinates.
- typedef float currentTime_t
- typedef float ObjectDistance

  Distance of an object from a detecting radar sensor.
- typedef float ObjectVelocity
- typedef int DataPacket_t [255]

Enumerations

- enum VehicleType { MOTORCYCLE, CAR }

  Enum for vehicle type.

6.8.1 Typedef Documentation

6.8.1.1 typedef float Coordinate_t

GPS coordinates.

Definition at line 22 of file MotorcycleAwarenessSystemTypes.hpp.

6.8.1.2 typedef float currentTime_t

Current time for a pair of GPS coordinates

Definition at line 23 of file MotorcycleAwarenessSystemTypes.hpp.

6.8.1.3 typedef int DataPacket_t[255]

Data packet for the V2V communication

Definition at line 48 of file MotorcycleAwarenessSystemTypes.hpp.

6.8.1.4 typedef float ObjectDistance

Distance of an object from a detecting radar sensor.

Definition at line 32 of file MotorcycleAwarenessSystemTypes.hpp.

6.8.1.5 typedef float ObjectVelocity

Velocity of an object detected by a radar sensor

Definition at line 33 of file MotorcycleAwarenessSystemTypes.hpp.

6.8.2 Enumeration Type Documentation

6.8.2.1 enum VehicleType

Enum for vehicle type.

Enumerator

MOTORCYCLE

CAR

Definition at line 8 of file MotorcycleAwarenessSystemTypes.hpp.

```
00009 {
00010     MOTORCYCLE,
00011     CAR
00012 };
```

6.9 MotorcycleAwarenessSystemTypes.hpp

```
00001 #ifndef MOTORCYCLEAWARENESSSYSTEMTYPES_HPP
00002 #define MOTORCYCLEAWARENESSSYSTEMTYPES_HPP
00003
00004 #include <list>
00005 #include <map>
00006
00008 enum VehicleType
00009 {
```

```
00010     MOTORCYCLE,
00011     CAR
00012 };
00013
00016 struct TurnSignal_t
00017 {
00018     bool isRightBlinkerOn;
00019     bool isLeftBlinkerOn;
00020 };
00021
00022 typedef float Coordinate_t;
00023 typedef float currentTime_t;
00024 struct GpsSignal_t
00026 {
00027     Coordinate_t x;
00028     Coordinate_t y;
00029     currentTime_t currentTime;
00030 };
00031
00032 typedef float ObjectDistance;
00033 typedef float ObjectVelocity;
00034 struct RadarSignal_t
00036 {
00037     ObjectDistance objectDistance;
00038     ObjectVelocity objectVelocity;
00039 };
00040
00042 struct BlueToothMessage_t
00043 {
00044     bool isHazardPresent;
00045     unsigned int dataBuffer[255];
00046 };
00047
00048 typedef int DataPacket_t[255];
00049 struct MotorCycleLocation_t
00051 {
00052     DataPacket_t dataPacket;
00053 };
00054 #endif
```

Index