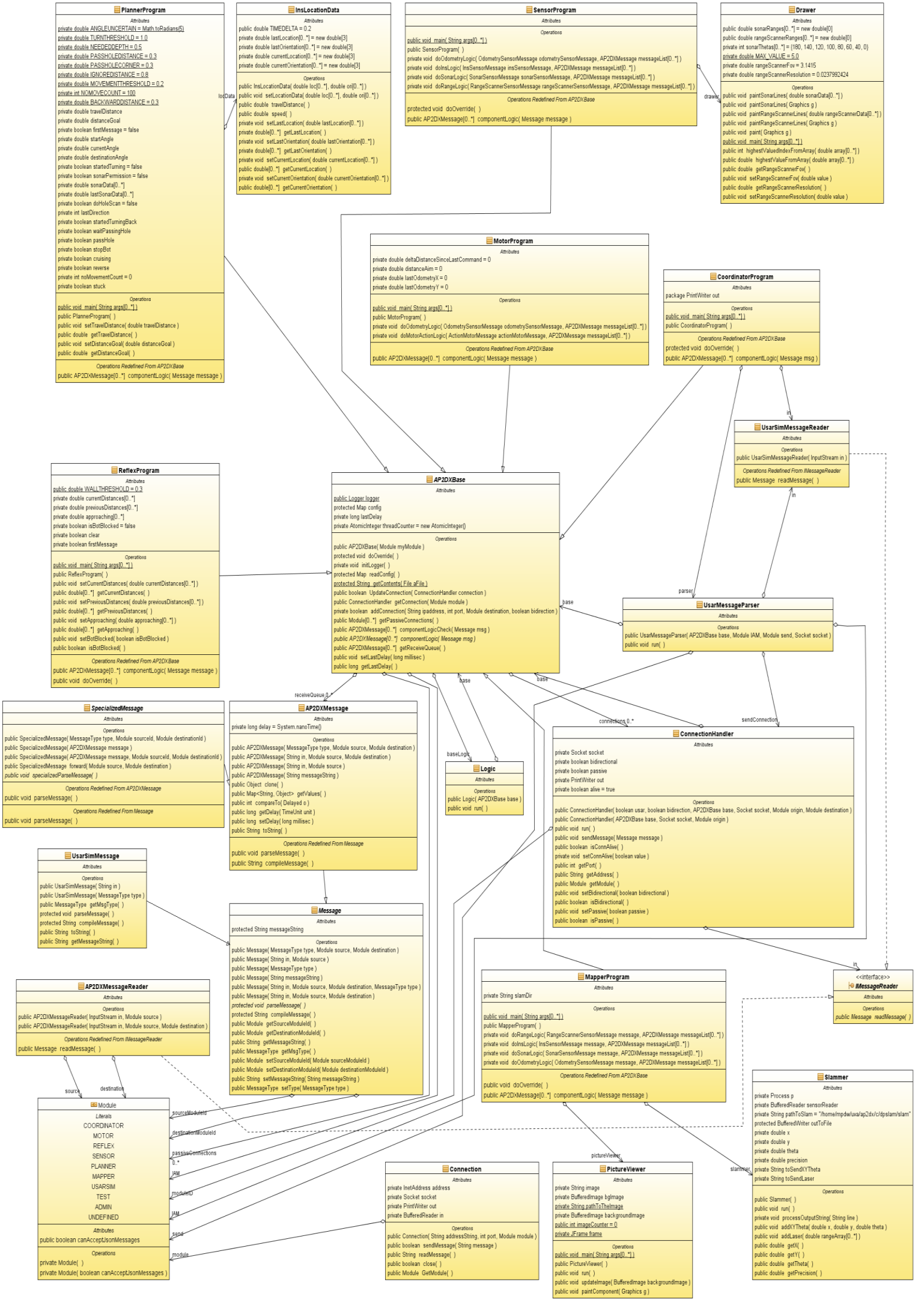
PlannerProgram
Attributes
private double ANGLEUNCERTAIN = Math.toRadians(5)
private double TURNTHRESHOLD = 1.0
private double NEEDEDDEPTH = 0.5
private double PASSHOLEDISTANCE = 0.3
private double PASSHOLECORNER = 0.3
private double IGNOREDISTANCE = 0.8
private double MOVEMENTTHRESHOLD = 0.2
private int NOMOVECOUNT = 100
private double BACKWARDDISTANCE = 0.3
private double travelDistance
private double distanceGoal
private boolean firstMessage = false
private double startAngle
private double currentAngle
private double destinationAngle
private boolean startedTurning = false
private boolean sonarPermission = false
private double sonarData[0..*]
private double lastSonarData[0..*]
private boolean doHoleScan = false
private int lastDirection
private boolean startedTurningBack
private boolean waitPassingHole
private boolean passHole
private boolean stopBot
private boolean cruising
private boolean reverse
private int noMovementCount = 0
private boolean stuck
Operations
public void main( String args[0..*] )
public PlannerProgram( )
public void setTravelDistance( double travelDistance )
public double getTravelDistance( )
public void setDistanceGoal( double distanceGoal )
public double getDistanceGoal( )
Operations Redefined From AP2DXBase
public AP2DXMessage[0..*] componentLogic( Message message )
InsLocationData
Attributes
public double TIMEDELTA = 0.2
private double lastLocation[0..*] = new double[3]
private double lastOrientation[0..*] = new double[3]
private double currentLocation[0..*] = new double[3]
private double currentOrientation[0..*] = new double[3]
Operations
public InsLocationData( double loc[0..*], double ori[0..*] )
public void setLocationData( double loc[0..*], double ori[0..*] )
public double travelDistance( )
public double speed( )
private void setLastLocation( double lastLocation[0..*] )
private double[0..*] getLastLocation( )
private void setLastOrientation( double lastOrientation[0..*] )
private double[0..*] getLastOrientation( )
private void setCurrentLocation( double currentLocation[0..*] )
public double[0..*] getCurrentLocation( )
private void setCurrentOrientation( double currentOrientation[0..*] )
public double[0..*] getCurrentOrientation( )
locData
SensorProgram
Attributes
Operations
public void main( String args[0..*] )
public SensorProgram( )
private void doOdometryLogic( OdometrySensorMessage odometrySensorMessage, AP2DXMessage messageList[0..*] )
private void doInsLogic( InsSensorMessage insSensorMessage, AP2DXMessage messageList[0..*] )
private void doSonarLogic( SonarSensorMessage sonarSensorMessage, AP2DXMessage messageList[0..*] )
private void doRangeLogic( RangeScannerSensorMessage rangeScannerSensorMessage, AP2DXMessage messageList[0..*] )
Operations Redefined From AP2DXBase
protected void doOverride( )
public AP2DXMessage[0..*] componentLogic( Message message )
Drawer
Attributes
public double sonarRanges[0..*] = new double[0]
public double rangeScannerRanges[0..*] = new double[0]
private int sonarThetas[0..*] = {180, 140, 120, 100, 80, 60, 40, 0}
private double MAX_VALUE = 5.0
private double rangeScannerFov = 3.1415
private double rangeScannerResolution = 0.0237992424
Operations
public void paintSonarLines( double sonarData[0..*] )
public void paintSonarLines( Graphics g )
public void paintRangeScannerLines( double rangeScannerData[0..*] )
public void paintRangeScannerLines( Graphics g )
public void paint( Graphics g )
public void main( String args[0..*] )
public int highestValuedIndexFromArray( double array[0..*] )
public double highestValueFromArray( double array[0..*] )
public double getRangeScannerFov( )
public void setRangeScannerFov( double value )
public double getRangeScannerResolution( )
public void setRangeScannerResolution( double value )
drawer
MotorProgram
Attributes
private double deltaDistanceSinceLastCommand = 0
private double distanceAim = 0
private double lastOdometryX = 0
private double lastOdometryY = 0
Operations
public void main( String args[0..*] )
public MotorProgram( )
private void doOdometryLogic( OdometrySensorMessage odometrySensorMessage, AP2DXMessage messageList[0..*] )
private void doMotorActionLogic( ActionMotorMessage actionMotorMessage, AP2DXMessage messageList[0..*] )
Operations Redefined From AP2DXBase
public AP2DXMessage[0..*] componentLogic( Message message )
CoordinatorProgram
Attributes
package PrintWriter out
Operations
public void main( String args[0..*] )
public CoordinatorProgram( )
Operations Redefined From AP2DXBase
protected void doOverride( )
public AP2DXMessage[0..*] componentLogic( Message msg )
in
UsarSimMessageReader
Attributes
Operations
public UsarSimMessageReader( InputStream in )
Operations Redefined From IMessageReader
public Message readMessage( )
in
ReflexProgram
Attributes
public double WALLTHRESHOLD = 0.3
private double currentDistances[0..*]
private double previousDistances[0..*]
private double approaching[0..*]
private boolean isBotBlocked = false
private boolean clear
private boolean firstMessage
Operations
public void main( String args[0..*] )
public ReflexProgram( )
public void setCurrentDistances( double currentDistances[0..*] )
public double[0..*] getCurrentDistances( )
public void setPreviousDistances( double previousDistances[0..*] )
public double[0..*] getPreviousDistances( )
public void setApproaching( double approaching[0..*] )
public double[0..*] getApproaching( )
public void setBotBlocked( boolean isBotBlocked )
public boolean isBotBlocked( )
Operations Redefined From AP2DXBase
public AP2DXMessage[0..*] componentLogic( Message message )
public void doOverride( )
AP2DXBase
Attributes
public Logger logger
protected Map config
private long lastDelay
private AtomicInteger threadCounter = new AtomicInteger()
Operations
public AP2DXBase( Module myModule )
protected void doOverride( )
private void initLogger( )
protected Map readConfig( )
protected String getContents( File aFile )
public boolean UpdateConnection( ConnectionHandler connection )
public ConnectionHandler getConnection( Module module )
private boolean addConnection( String ipaddress, int port, Module destination, boolean bidirection )
public Module[0..*] getPassiveConnections( )
public AP2DXMessage[0..*] componentLogicCheck( Message msg )
public AP2DXMessage[0..*] componentLogic( Message msg )
public AP2DXMessage[0..*] getReceiveQueue( )
public void setLastDelay( long millisec )
public long getLastDelay( )
base
parser
UsarMessageParser
Attributes
Operations
public UsarMessageParser( AP2DXBase base, Module IAM, Module send, Socket socket )
public void run( )
receiveQueue 0..*
base
base
base
connections 0..*
sendConnection
SpecializedMessage
Attributes
Operations
public SpecializedMessage( MessageType type, Module sourceId, Module destinationId )
public SpecializedMessage( AP2DXMessage message )
public SpecializedMessage( AP2DXMessage message, Module sourceId, Module destinationId )
public SpecializedMessage forward( Module source, Module destination )
public void specializedParseMessage( )
Operations Redefined From AP2DXMessage
public void parseMessage( )
Operations Redefined From Message
public void parseMessage( )
AP2DXMessage
Attributes
private long delay = System.nanoTime()
Operations
public AP2DXMessage( MessageType type, Module source, Module destination )
public AP2DXMessage( String in, Module source, Module destination )
public AP2DXMessage( String in, Module source )
public AP2DXMessage( String messageString )
public Object clone( )
public Map<String, Object> getValues( )
public int compareTo( Delayed o )
public long getDelay( TimeUnit unit )
public long setDelay( long millisec )
public String toString( )
Operations Redefined From Message
public void parseMessage( )
public String compileMessage( )
baseLogic
Logic
Attributes
Operations
public Logic( AP2DXBase base )
public void run( )
ConnectionHandler
Attributes
private Socket socket
private boolean bidirectional
private boolean passive
private PrintWriter out
private boolean alive = true
Operations
public ConnectionHandler( boolean usar, boolean bidirection, AP2DXBase base, Socket socket, Module origin, Module destination )
public ConnectionHandler( AP2DXBase base, Socket socket, Module origin )
public void run( )
public void sendMessage( Message message )
public boolean isConnAlive( )
private void setConnAlive( boolean value )
public int getPort( )
public String getAddress( )
public Module getModule( )
public void setBidirectional( boolean bidirectional )
public boolean isBidirectional( )
public void setPassive( boolean passive )
public boolean isPassive( )
UsarSimMessage
Attributes
Operations
public UsarSimMessage( String in )
public UsarSimMessage( MessageType type )
public MessageType getMsgType( )
protected void parseMessage( )
protected String compileMessage( )
public String toString( )
public String getMessageString( )
Message
Attributes
protected String messageString
Operations
public Message( MessageType type, Module source, Module destination )
public Message( String in, Module source )
public Message( MessageType type )
public Message( String messageString )
public Message( String in, Module source, Module destination, MessageType type )
public Message( String in, Module source, Module destination )
protected void parseMessage( )
protected String compileMessage( )
public Module getSourceModuleId( )
public Module getDestinationModuleId( )
public String getMessageString( )
public MessageType getMsgType( )
public Module setSourceModuleId( Module sourceModuleId )
public Module setDestinationModuleId( Module destinationModuleId )
public String setMessageString( String messageString )
public MessageType setType( MessageType type )
in
<<interface>>
IMessageReader
Attributes
Operations
public Message readMessage( )
MapperProgram
Attributes
private String slamDir
Operations
public void main( String args[0..*] )
public MapperProgram( )
private void doRangeLogic( RangeScannerSensorMessage message, AP2DXMessage messageList[0..*] )
private void doInsLogic( InsSensorMessage message, AP2DXMessage messageList[0..*] )
private void doSonarLogic( SonarSensorMessage message, AP2DXMessage messageList[0..*] )
private void doOdometryLogic( OdometrySensorMessage message, AP2DXMessage messageList[0..*] )
Operations Redefined From AP2DXBase
public void doOverride( )
public AP2DXMessage[0..*] componentLogic( Message message )
AP2DXMessageReader
Attributes
Operations
public AP2DXMessageReader( InputStream in, Module source )
public AP2DXMessageReader( InputStream in, Module source, Module destination )
Operations Redefined From IMessageReader
public Message readMessage( )
source
destination
Module
Literals
COORDINATOR
MOTOR
REFLEX
SENSOR
PLANNER
MAPPER
USARSIM
TEST
ADMIN
UNDEFINED
Attributes
public boolean canAcceptJsonMessages
Operations
private Module( )
private Module( boolean canAcceptJsonMessages )
sourceModuleId
destinationModuleId
passiveConnections 0..*
IAM
moduleID
IAM
send
module
Slammer
Attributes
private Process p
private BufferedReader sensorReader
private String pathToSlam = "/home/impdw/uva/ap2dx/c/dpslam/slam"
protected BufferedWriter outToFile
private double x
private double y
private double theta
private double precision
private String toSendXYTheta
private String toSendLaser
Operations
public Slammer( )
public void run( )
private void processOutputString( String line )
public void addXYTheta( double x, double y, double theta )
public void addLaser( double rangeArray[0..*] )
public double getX( )
public double getY( )
public double getTheta( )
public double getPrecision( )
slammer
Connection
Attributes
private InetAddress address
private Socket socket
private PrintWriter out
private BufferedReader in
Operations
public Connection( String addressString, int port, Module module )
public boolean sendMessage( String message )
public String readMessage( )
public boolean close( )
public Module getModule( )
pictureViewer
PictureViewer
Attributes
private String image
private BufferedImage bgImage
private String pathToTheImage
private BufferedImage backgroundImage
public int imageCounter = 0
private JFrame frame
Operations
public void main( String args[0..*] )
public PictureViewer( )
public void run( )
public void updateImage( BufferedImage backgroundImage )
public void paintComponent( Graphics g )