PennState
Applied Research Laboratory

Final Report for EOD Robotics Future Technology Assessment, Integration, and Analysis for the Office of Naval Research Department of Defense Explosive Ordnance Disposal (EOD) Applied Research Program

Gazebo/ROS Test Bed Documentation

Kurt Hacker, Jessie Pentzer, Simon Miller, Robert Peterson

Technical Memorandum
File No.: TM 19-035
Date: June 27, 2019

Applied Research Laboratory
P.O. Box 30
State College, PA 16804

Contracted by: Office of Naval Research
875 North Randolph Street
Arlington, VA 22203-1995

Title: Gazebo/ROS Test Bed Documentation

Contract No.: N00014-16-1-2673

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
6/27/2019	Final Technical Report	01 June 2016 - 30 June 2019

4. TITLE AND SUBTITLE
EOD Robotics Future Technology Assessment, Integration, and Analysis for the Office of Naval Research Department of Defense Explosive Ordance Disposal (EOD) Applied Research Program

5a. CONTRACT NUMBER

5b. GRANT NUMBER
N00014-16-1-2673

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kurt Hacker

5d. PROJECT NUMBER
24429

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Pennsylvania State University Applied Research Labotatory
Office of Sponsored Programs
110 Technology Center Building
University Park, PA 16802-7000

8. PERFORMING ORGANIZATION REPORT NUMBER
TM 19-035

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
875 North Randolph Street
Arlington, VA 22203-1995

10. SPONSOR/MONITOR'S ACRONYM(S)
ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this project, Penn State University Applied Research Laboratory (PSU/ARL) developed the Gazebo/ROS Testbed to enable the Advanced Explosive Ordance Robotics System (AEODRS) testbed and its associated Capability Modules to utilize and interface with the open source Gazebo physics simulation environment. Using Gazebo provides many benefits to the AEODRS test bed. Gazebo is an open source software project widely used within academia and has a proven track record of robot system simulation in virtual environments through many government sponsored research programs such as the DARPA Robotics Challenge. Gazebo provides built-in support for sensor and manipulator simulation, both of which are necessary for the AEODRS test bed. In this work, the Robot Operating System (ROS) is used to bridge the AEODRS JAUS communication between CMs with the Gazebo simulator.

15. SUBJECT TERMS
EOD, Robotics, Simulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	SAR	22	Kurt Hacker
U	U	U			19b. TELEPONE NUMBER (*Include area code*) (814) 867-3802

PennState
Applied Research
Laboratory

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

Gazebo/ROS Test Bed Documentation

Jesse Pentzer (jlp5573@arl.psu.edu)

Simon Miller

Robert Peterson

Kurt Hacker

June 27, 2019

UNCLASSIFIED

PennState
Applied Research
Laboratory

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

Table of Contents

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

List of Figures

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

1 Introduction

The AEODRS test bed is a set of software designed to emulate the capability modules (CMs) of a robot designed to conform to the Advanced Explosive Ordnance Disposal Robotic System (AEODRS) architecture standards. The surrogate CMs present a communication interface identical to that of an actual hardware component and are useful for system development and hardware-in-the-loop testing of production CMs. Within the AEODRS architecture, a complete robot is comprised of many CMs that perform a specific function and communicate with each other using the Joint Architecture for Unmanned Systems (JAUS) standard [1]. The simulated CMs are able to provide meaningful sensor feedback to the operator by interfacing with a physics model. The goal of this project was to develop a set of CM surrogates that interface with the open source Gazebo physics simulation environment [2].

Using Gazebo provides many benefits to the AEODRS test bed. Gazebo is an open source software project widely used within academia and has a proven track record of robot system simulation in virtual environments through many government sponsored research programs such as the DARPA Robotics Challenge. Using open source tools removes barriers to project entry and eases the process whereby academic and industry research is integrated into a developed system. Gazebo provides built-in support for sensor and manipulator simulation, both of which are necessary for the AEODRS test bed. In this work, the Robot Operating System (ROS) is used to bridge the AEODRS JAUS communication between CMs with the Gazebo simulator. ROS is a middleware software package that enables robot software developers to easily connect sensors, controllers, actuators, user interfaces, and applied algorithms using network communications on a single computer or a network of computers [3]. In this project, ROS is utilized to provide communications between the CMs and Gazebo and also for closed loop control of the simulated manipulator joints.

The remaining sections in this report document the Gazebo/ROS test bed architecture, how the test bed is started, and which files within the test bed source code were modified to create the test bed.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

2 Gazebo/ROS Test Bed Architecture

This section will provide a general overview of the Gazebo/ROS test bed architecture and the different software components it is composed of. The Gazebo/ROS test bed was developed on a computer running Ubuntu 16.04 with ROS Kinetic and Gazebo 7. If newer versions of ROS are used to run the test bed, the version of Gazebo recommended on the ROS wiki should be used with it. The test bed itself consists of many executable programs that are run concurrently, and the programs communicate with Gazebo using ROS and with the Multi-Robot Operator Control Unit (MOCU) software using JAUS. A diagram of the test bed architecture is shown in Fig. 1. The goal of this work was to simulate a robot modeled after the AEODRS Increment 1 robot, and modeling a different robot might produce an architecture different than that shown in Fig. 1.

The intra-subsystem network connects CMs within a single robot together. Communication between a robot and any outside entity, most likely an operator control unit (OCU), is managed by CM-Master. The CM-Master software used in the Gazebo/ROS test bed is the same software used in the standard AEODRS test bed distributed by JHU-APL. CM-Master monitors the inter-subsystem network for traffic addressed to CMs on the robot and is responsible for forwarding traffic to the correct CM on the robot and from CMs back to the OCU.

Returning to Fig. 1, the majority of software development occurred within the "AEODRS CM Surrogates" block. The starting point for software development was a collection of skeleton CM code provided by JHU-APL. Originally this software was automatically generated using the JAUS Toolset software [4]. JHU-APL did a significant amount of work to take the generated code and build a working collection of JAUS surrogates with no actual functionality. It is left to the final developer to add the required code that interprets received communications, interfaces with hardware (or a physics model), and provides the correct output data as requested. The focal point for JAUS communication is the JTS Node Manager software. This is provided by the JAUS Toolset and manages all network communications for all JAUS entities on a single computer. The surrogate CM programs register the AEODRS messages they support with JTS Node Manager, and Node Manager subsequently forwards received network traffic to the appropriate CMs. When a message is received, a call-

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

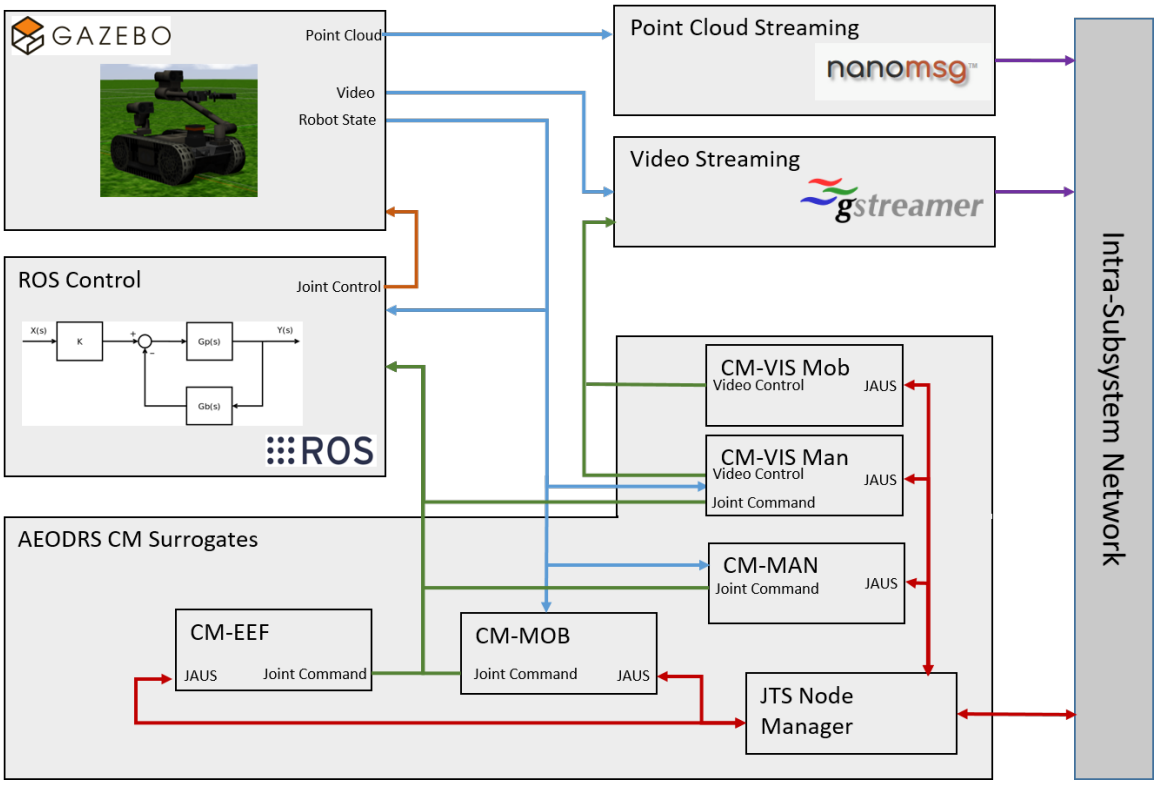

Figure 1: Architecture diagram of Gazebo/ROS test bed.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

back function within a CM is called, and this is where the user must insert software to properly handle the message and create a reply message if required.

The "ROS Control" section of Fig. 1 is responsible for performing closed loop control of all movable joints on the robot. Joint motion commands transmitted from MOCU are received by CM-EEF, CM-MOB, and CM-MAN. The CMs parse the joint motion messages and publish the commands to ROS topics corresponding to each joint. ROS Control subscribes to the joint command topics as well as joint state feedback produced by Gazebo. After comparing the commands and current joint states, ROS Control publishes joint torque commands that are fed to Gazebo. Gazebo applies the torques to the proper joints and simulates the motion of the robot. The CMs also subscribe to the joint states published by Gazebo and send information on the current state of the robot back to MOCU when requested.

There are two different Visual Sensor capability modules on the robot: one on the manipulator and another mounted on the robot chassis. This matches the Increment 1 robot design. Gazebo simulates cameras at both locations on the robot and publishes image streams on ROS topics. The CM-VIS modules receive requests for video streams from MOCU and create ROS service requests to activate a video streaming node developed for the ROS/Gazebo test bed. AEODRS requires that video streams be encoded using the H.264 codec and sent over the network connection using the Real-Time Transfer Protocol (RTP) [5]. The open source multimedia framework GStreamer is used to both encode the simulated camera images and stream them to MOCU using RTP [6]. The "Video Streaming" module provides a ROS service to start and stop video streams. The service call includes the video topic that should be streamed, the end point of the stream, and the size of the video frames.

The "Point Cloud Streaming" module in Fig. 1 is not an AEODRS component but was required to transport point cloud data from Gazebo to the MIT manipulation planning software. Integrating the MIT manipulation planning software with MOCU is the other focus of this work and is discussed in additional documentation developed during this project. On an actual robot, it is likely that the operator control unit would have some representation of the world surrounding the robot, commonly referred to as a world model. We did not have such a representation created within MOCU for the Gazebo/ROS test bed, so we created a point cloud streaming ROS node as a workaround. Gazebo publishes the output of a simulated point cloud LiDAR sensor

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

on a ROS topic. The point cloud streaming node subscribes to this topic, repackages the data, and transmits it over the network using the nanomsg library [7]. Nanomsg was used because it is available for Windows and Linux, and a program that is very similar to the point cloud node had already developed for another project. A different network messaging library could easily be installed in place of it.

Finally, the "Gazebo" module in Fig. 1 contains the physics simulation. A world description file is created to contain all of the obstacles the robot will interact with in the virtual world, and the robot model is loaded dynamically at run time when the CM surrogates are started.

3 Software Layout

All of the Gazebo/ROS test bed software is contained within a single ROS catkin workspace. Catkin is the default build system in ROS [8]. The work flow is very similar to CMake, but catkin provides support for finding other ROS packages through CMake macros and Python scripts. The advantage of placing everything inside a catkin workspace is that non-ROS libraries are easily built and included by dependent projects without system installation. Fig. 2 shows the folders within the catkin workspace of the Gazebo/ROS test bed. The folders highlighted in green contain source code that a developer will most likely modify to continue test bed development, the folders highlighted in yellow contain source code that will infrequently require modification, and the folders highlighted in red contain source code that should not need to be edited.

- **3rd_party**: Contains source code for a Gazebo plugin to add the mimic joint functionality that exists in ROS.
- **cm_eef**: Contains a ROS node that starts the CM End Effector surrogate software.
- **cm_man**: Contains a ROS node that starts the CM Manipulator surrogate software.
- **cm_mob**: Contains a ROS node that starts the CM Mobility surrogate software.
- **cm_vis**: Contains a ROS node that starts the CM Visual Sensor surrogate

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

software.

- **inc1_control**: Contains ROS launch files and configuration software for starting the Increment 1 robot simulation.
- **inc1_description**: Contains ROS robot description files and Gazebo world description files for the Increment 1 robot simulation.
- **inc1_pprm_control**: Contains ROS launch files and configuration software for starting the Increment 1 PPRM robot simulation.
- **inc1_pprm_description**: Contains ROS robot description files and Gazebo world description files for the Increment 1 PPRM robot simulation.
- **jtscommon**: Contains JAUS Toolset libraries developed by JHU-APL.
- **jts_components**: Contains the source code for each JAUS component generated from the JAUS Toolset software.
- **jtsextension**: Contains software developed by JHU-APL to handle common AEODRS/JAUS requirements.
- **jtsnodemanager**: Contains the source code for the JTS Node Manager software provided with JAUS Toolset.
- **jts_services**: Contains the source code for each JAUS service generated from the JAUS Toolset software.
- **pc_sender**: Contains the ROS node source code for streaming point cloud data across the network.
- **rtp_image_server**: Contains the ROS service source code for streaming images across the network.

4 Robot and World Description

This section will describe the robot model description format and the world model created for Gazebo.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

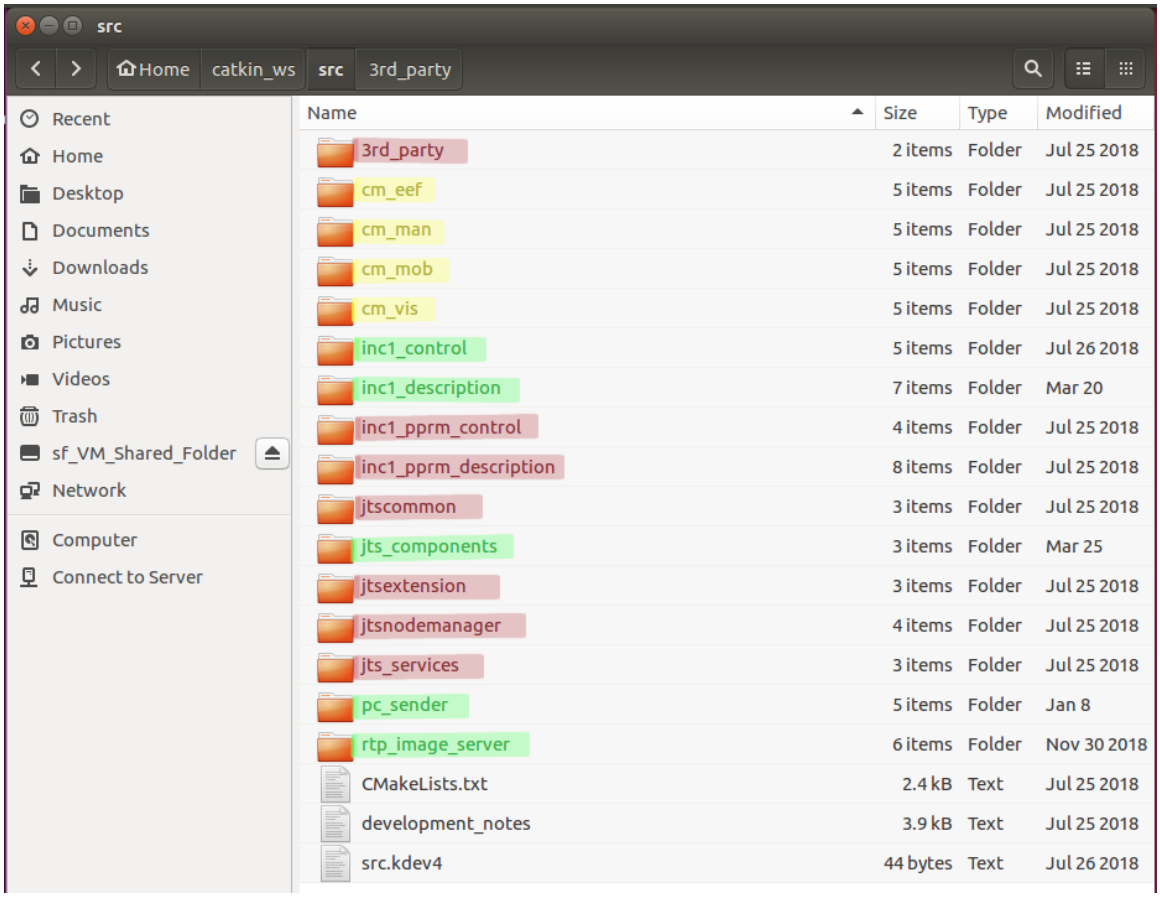

Figure 2: Folders within the catkin workspace of the Gazebo/ROS testbed.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

4.1 Robot URDF

The Increment 1 robot model is stored in Universal Robot Description Format (URDF) files. URDF files are commonly used within the ROS ecosystem to represent robotic systems, and there is also a macro language named xacro designed to increase URDF file readability and maintainability [9]. The Increment 1 robot URDF files are located in the directory `inc1_description/urdf` and the file currently loaded is `inc1_lidar.xacro`. The mesh files and texture images used to produce an accurate representation of the Increment 1 robot are stored in `inc1_description/meshes` and are referenced within the URDF files. The first step in simulating a new robot in the Gazebo/ROS test bed is to develop a new URDF describing the robot and generate meshes to represent the robot visually. It is important that the joints comprising the robots mobility mechanism and/or manipulators be named consistently and descriptively because the names will be used throughout the test bed to connect controllers and JAUS message fields to the correct joint. In the Increment 1 URDF, the chassis is named `base_link`, the first manipulator joint is named `base_link_to_arm_link_1`, the second manipulator joint is named `arm_link_1_to_arm_link_2`, and so on down the manipulator chain. An image of the Increment 1 robot simulated within Gazebo is shown in Fig. 3.

4.2 World Model

The world model file loaded when the Gazebo/ROS test bed starts is located in the directory `inc1_description/worlds` and is named `Demo_World.world`. The format of a Gazebo world file is described on the OSRF SDFormat web page [10]. The world file currently used was not created by hand; instead it was created using the Gazebo user interface and saved directly to a world file. The tutorial named "Building a World" on the Gazebo website was followed to create the demo world [2]. The Gazebo Model Database provides many models that may be easily inserted into the test world. Internet is required to download the models the first time they are inserted, but it is not required to load the world again once it is saved. Note that the robot is not saved in the world file. It is instead loaded dynamically when ROS starts the CM surrogates.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

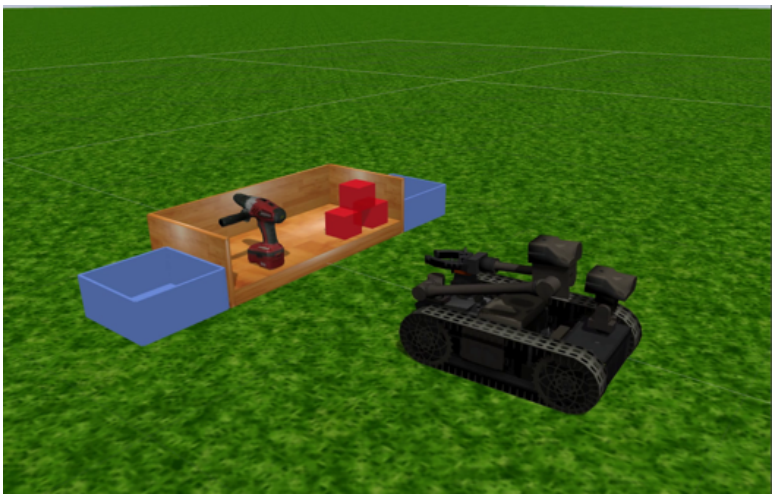

Figure 3: Increment 1 robot in Gazebo simulation.

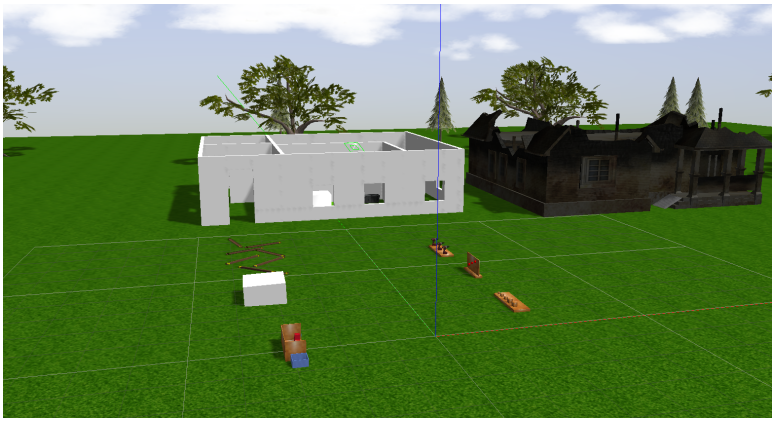

Figure 4: Test bed Gazebo test world.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

5 Operation of Gazebo/ROS Test Bed

This section will describe the startup procedure for the Gazebo/ROS test bed and provide an explanation of the ROS launch files that start the simulation software.

The networks, CM-MAS, and MOCU test bed computers should be setup as described in the AEODRS test bed manual. It is recommended that the test bed be started in the order:

1. ROS launch file for Gazebo/ROS surrogates and supporting software
2. CM-MAS
3. MOCU

The ROS/Gazebo surrogates and supporting software modules are started using the roslaunch command

```
roslaunch inc1_control main.launch
```

in a terminal that has sourced the setup file for the Gazebo/ROS catkin workspace. CM-MAS and MOCU are started as described in the AEODRS test bed manual.

5.1 File Description: inc1_control/launch/main.launch

The first step in this launch file is to set the variable `robot_description` to the path of the Increment 1 URDF description file previously discussed. Next, a supplemental launch file is included named `inc1_gazebo_world.launch` that will be described next. This launch file creates the Gazebo simulation world, and one of the arguments passed to this launch file is the path to the world definition file `Demo_World.world`. Following creation of the simulation world, the robot model is loaded into the Gazebo simulation using the `spawn_model` node within the `gazebo_ros` package.

After the robot is spawned, a series of additional launch files are included:

- `inc1_control/launch/inc1_control.launch`: Launch file to create ROS closed loop controllers for robot joints.
- `jtsnodemanager/launch/main.launch`: Launch file to start the JTS node manager.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

- `cm_mob/launch/main.launch`: Launch file to start the CM-MOB surrogate.
- `cm_man/launch/main.launch`: Launch file to start the CM-MAN surrogate.
- `cm_eef/launch/main.launch`: Launch file to start the CM-EEF surrogate.
- `cm_vis/launch/main.launch`: Launch file to start the CM-VIS surrogate. Note that this launch file is included twice with different arguments to create two CM-VIS surrogates at different locations on the robot.

The `joint_name_list` and `position_controller_topics` arguments passed to the launch files inform the node as to which joints it controls. In the case of CM-MAN, the lists of joints must be in the order of robot base to end effector because that is the order in which joint commands are packaged in JAUS control messages.

The final three entries in this launch file start the ROS gmapping node to create maps from range sensor output, the RTP image server for streaming video, and the point cloud data network streaming node.

5.2 File Description: inc1_description/launch/inc1_gazebo_world.launch

This launch file is a copy of the example `empty_world.launch` test file installed with the "gazebo_ros" package. `empty_world.launch` by itself starts a Gazebo simulation with a completely empty world. By passing in arguments to overide the `world_name` variable, the test bed demonstration world can be loaded instead. Making a copy of `empty_world.launch` also allows default parameters to be changed as needed during test bed development.

5.3 File Description: inc1_control/launch/inc1_control.launch

The `inc1_control.launch` file loads joint controllers by calling the "spawner" node from within the "controller_manager" package. This launch file first loads the configuration file `/inc1_control/config/inc1_control.yaml`, which contains names, controller types, joint assignments, and PID gains for controllers attached to all movable joints on the robot. After loading the configuration file, the controllers are

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

spawned and all of the controller names are passed to the spawner node. The final portion of this launch file starts a "robot_state_publisher" node that subscribes to joint state data published by Gazebo and publishes ROS TF transforms useful for visualizing the robot.

5.4 File Description: jtsnodemanager/launch/main.launch

The `main.launch` file within the "jtsnodemanager" package starts an instance of the JTS Node Manager software. Only one instance of Node Manager can be run at one time because the program must gain exclusive control over the JAUS Ethernet port, number 3794.

5.5 File Description: Surrogate CM main.launch files

All of the `main.launch` files for the surrogate CMs have identical structures with the only difference being which CM surrogate ROS node is launched. It is important to note that in `inc1_control/launch/main.launch` the list of topics the node should subscribe to and the list of topics the node should publish commands to are provided as arguments when the surrogate `main.launch` file is included. These lists of topics must be in the same order.

6 CM-MOB Description

CM-MOB contains the chassis and drive elements of the robot, and it provides the interface by which the motion of the robot is controlled. During initial startup, it is necessary to set constant physical parameters of the capability module. Examples of how this is done are given in the file

`cm_mob/src/main.cpp`

The files that must be edited to handle incoming JAUS messages and interface with the Gazebo simulator are located in

`jts_components/Components/Mobility_102/include/`

and will be described in the following subsections.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

6.1 File Description: AEODRS_PrimitiveDriver_ReceiveFSM.h

The Primitive Driver JAUS component is the most basic method of controlling a joint through commanded effort. The function `setWrenchEffortAction` within this file is called whenever a command to set the driving effort is received by the CM-MOB Primitive Driver component. The effort command is received as a linear propulsive effort command and a rotational effort command. The software in this function mixes the linear effort and rotational effort into left and right track effort commands, which are then published to the appropriate ROS controller topics.

6.2 File Description: AEODRS_VelocityStateDriver_ReceiveFSM.h

The Velocity State Driver JAUS component is used to control the velocity of a joint. The function `setCurrentVelocityCommandAction` within this file is called whenever a command to set the current velocity of the vehicle is received by the CM-MOB Velocity State Driver component. The velocity command is currently transformed into an effort command and published to the appropriate ROS topics. This was done because it is difficult to switch the corresponding ROS controller from an effort controller to a velocity controller during operation.

7 CM-EEF Description

CM-EEF is used to control the end effector attached to the end of the manipulator while also providing information on the state of the end effector in real time. During initial startup, it is necessary to set constant physical parameters of the capability module. Examples of how this is done are given in the file

`cm_eef/src/main.cpp`

The files that must be edited to handle incoming JAUS messages and interface with the Gazebo simulator are located in

`jts_components/Components/EndEffector_105/include`

and will be described in the following subsections.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

7.1 File Description: AEODRS_PrimitiveEndEffector_ReceiveFSM.h

The Primitive End Effector JAUS component is used to control the motion of end effector joints. The function `setEndEffectorEffortAction` within this file is called whenever a command to set the end effector motion effort is received. To interface with the ROS controller architecture, the effort command is applied as a commanded change in position. Stored end effector joint positions are incremented a small amount based on the direction of the commanded effort and then published as set points for the ROS controllers moving the end effector joints.

8 CM-MAN Description

CM-MAN is responsible for the control of a single manipulator and also provides feedback to MOCU on the structure of the manipulator (joint length, joint connection angles, etc) and the state of the manipulator joints. It is possible for a robot to have multiple manipulators, and thus multiple CM-MAN systems, but the Increment 1 robot has only a single manipulator. During initial startup, it is necessary to set constant physical parameters of the capability module. Examples of how this is done are given in the file

`cm_man/src/main.cpp`

The files that must be edited to handle incoming JAUS messages and interface with the Gazebo simulator are located in

`jts_components/Components/Manipulator_104/include`

and will be described in the following subsections.

8.1 File Description: AEODRS_PrimitiveManipulator_ReceiveFSM.h

The Primitive Manipulator JAUS component is used to control the motion of the manipulator joints using commanded joint efforts. Within this file the function `setJointEffortsAction` is called whenever a command to set the manipulator joint

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

effort is received. To interface with the ROS controller architecture, the effort command is applied as a commanded change in position. Stored manipulator joint positions are incremented a small amount based on the direction of the commanded effort and then published as set points for the ROS controllers moving the manipulator joints. Within the Primitive Manipulator class constructor contained within this file, the orientation of the manipulator coordinate system and manipulator specifications are set so that they can be provided to requesting components as needed.

8.2 File Description:

AEODRS_ManipulatorJointPositionDriver_ReceiveFSM.h

The Manipulator Joint Position Driver JAUS component is used to control the motion of the manipulator joints using commanded joint positions. Within this file is the function `setJointPositionsAction` that is called whenever a command to set the manipulator joint position is received. Because joint position controllers are used within ROS to control the simulated manipulator, the commanded joint positions are directly published to the corresponding ROS control topics.

8.3 File Description:

AEODRS_ManipulatorJointPositionSensor_ReceiveFSM.h

The Manipulator Joint Position Sensor JAUS component is used to report manipulator joint position information to a requesting JAUS component. The function `Joint_State_Callback` in this file is called whenever Gazebo publishes updated joint position information to a ROS topic. The function stores the joint position information in a class member variable and then calls a function to inform the JAUS component state machine that new values are available.

8.4 File Description:

AEODRS_ManipulatorJointVelocitiesDriver_ReceiveFSM.h

The Manipulator Joint Velocities Driver JAUS component is used to control the motion of the manipulator using commanded joint velocities. Within this file is the `setJointVelocitiesAction` function that is called whenever a command to set the manipulator joint velocities is received. To interface with the ROS controller architecture, the velocity command is applied as a commanded change in position.

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

Stored manipulator joint positions are incremented a small amount based on the direction of the commanded velocity and then published as set points for the ROS controllers moving the manipulator joints.

9 CM-VIS Description

CM-VIS is responsible for the control of video sensors on the robot and also creates and destroys streams to transmit video feeds over the Ethernet network. During initial startup, it is necessary to set constant physical parameters of the capability module. Examples of how this is done are given in the file

`cm_vis/src/main.cpp`

The files that must be edited to handle incoming JAUS messages and interface with the Gazebo simulator are located in

`jts_components/Components/VisualSensor_106/include`

and will be described in the following subsections.

9.1 File Description: MyAEODRSStillImageSensor_ReceiveFSM.h

The Still Image Sensor JAUS component is used to obtain the capabilities of an imaging sensor on a robot, configure the sensor, and also obtain still image data from the sensor. The class constructor in this file initializes variables to describe the capabilities of the sensor so that this information can be provided to requesting components. There are functions within this file to update the sensor configuration when commanded and also to provide a still image when one is requested. Currently the only setting that is fully implemented is the image size. Other setting changes are stored but are not reflected in the Gazebo image sensor simulation.

9.2 File Description: MyAEODRSDigitalVideo_ReceiveFSM.h

The Digital Video JAUS component is used to report digital video sensor capabilities, configure the video sensor, and also setup video streams. The class constructor in

P.O. Box 30
State College, PA 16804-0030
(814)-865-6531

this file contains the initialization of the variables containing the digital video sensor capabilities so that they can be reported to requesting components. The function `setVideoEndpointAction` is called whenever a command to set a video endpoint IP address and port is received. When this occurs, a ROS service call is created containing all of the information regarding the video stream including the requested endpoint and video frame size. The `modifyDigitalVideoSensorStreamAction` function is called when a command to modify the video stream is received. The command can be to start, stop, or pause the video stream. A ROS service call is performed to execute the stream action.

References

[1] *JAUS Core Service Set*, Society of Automotive Engineers Std. AS5710A, Rev. 2015-04-24.

[2] Open Source Robotics Foundation, "Gazebo," http://gazebosim.org/, 2019.

[3] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, 2009.

[4] Neya Systems LLC, "JAUS toolset," http://jaustoolset.org/, 2019.

[5] Internet Engineering Task Force, "RTP: A transport protocol for real-time applications," https://tools.ietf.org/html/rfc3550, 2019.

[6] GStreamer, "GStreamer open source multimedia framework," https://gstreamer.freedesktop.org/, 2019.

[7] nanomsg, "nanomsg socket library," https://nanomsg.org/, 2019.

[8] Open Source Robotics Foundation, "Catkin package summary," http://wiki.ros.org/catkin, 2019.

[9] ——, "URDF: Universal robot description format," http://wiki.ros.org/urdf, 2019.

[10] ——, "SDFormat specification," http://sdformat.org/spec, 2019.