Generating a ROS to JAUS Bridge for an Autonomous Ground Vehicle

Patrick Morley
The University of Akron
pjm39@zips.uakron.edu

Alex Warren
University of Arizona
amwarren@email.arizona.edu

Ethan Rabb
Washington University in St. Louis
ethanrabb@gmail.com

Sean Whitsitt
University of Arizona
whitsitt@email.arizona.edu

Matt Bunting
University of Arizona
mosfet@email.arizona.edu

Jonathan Sprinkle
University of Arizona
sprinkle@ece.arizona.edu

ABSTRACT

Robotics, specifically robotic ground vehicles, are a rapidly growing field of interest in both academia and industry today. These systems tend to be developed with component based architectures that work across a standard TCP/IP network such as the Robot Operating System (ROS) and the Joint Architecture for Unmanned Systems (JAUS). Depending on the requirements of the project in question, engineers will default to using one standard or another for their purposes. For instance, JAUS tends to be used in the defense industry in the United states since it was developed by the US Department of Defense (DoD) while ROS offers more advantages to the open source community. As such, ROS has a large library of open source components available to run many different sensors and simulations off the shelf, without requiring the developer to spend valuable time designing low level drivers or physics environments. However, ROS and JAUS were not designed to be able to communicate between each other. This paper will detail the methods employed to bridge the gap between ROS and JAUS and to construct a code generator capable of building a customized node capable of communications with both standards. As an example scenario this paper will show how these bridges have been implemented to both run a physical autonomous vehicle (a modified Ford Hybrid Escape) and to handle the dynamics of that vehicle in simulation. Future work includes integrating this code generator in a larger project for modeling systems built jointly between ROS and JAUS.

1. INTRODUCTION

The complexity involved in autonomous vehicles projects makes them ideal implementation platforms for component-based systems. These projects require the integration of software that can manage sensing, control of vehicle actuation, and logging information for safety and debugging purposes. Component-based designs permit a functional decomposition of the tasks involved into atomic processes allowing them to communicate via message passing, abstracting even whether a set of tasks are operating on the same machine or across a network.

The Cognitive and Autonomous Testing Vehicle (CATVehicle) at the University of Arizona takes advantage of the component-based infrastructure of the Joint Architecture for Unmanned Systems (JAUS) [7]. Currently, the vehicle has components designed in JAUS for it that allow for steering and velocity control, Global Positioning System (GPS) and Inertial Navigation System (INS) path following, trajectory planning, and other non-sensor related tasks. With the exception of the GPS/INS component, the CATVehicle is driving with Dead Reckoning [4]. This unfortunately means that the vehicle is incapable of driving for long distances without accumulating a critical amount of error making operation unsafe. As an aside here, the reader should be aware that safety in autonomous vehicles is of paramount importance to the authors.

ROS (Robot Operating System) [6] is a widely used open-source architecture in robotics, capable of interaction with a significant number of available sensors, simulators, and programs. The availability of such code makes it ideal for code reuse in larger research projects allowing research scientists and engineers to concentrate more on the research aspects of the project than reimplementing trivial software packages and programs. As such, ROS is an ideal standard with which the CATVehicle can be integrated after overcoming the engineering challenges posed by integrating ROS and JAUS. This is the motivation behind integrating these two similar, yet disparate, standards. Assuming full integration, JAUS can provide the Department of Defense (DoD) specified foundation for the vehicle while ROS can provide the low level integration with advanced sensors for the vehicle.

1.1 Terms

While ROS and JAUS are both component based systems with (generally) a one to one correspondence between concepts, they do not use the same terminology. For the purposes of brevity and clarity in this paper, the terms used herein will be described and linked between the two standards in this subsection. These terms can be used interchangeably, and this paper will attempt to use the correct terminology in the differing contexts. The base level task or component in a ROS system is called a node while in JAUS it is called a component In order to pass data between components in the system ROS uses the term topic while JAUS uses the term message. In ROS, nodes must publish data in order for other nodes to access or subscribe to that data. In JAUS, components can send single messages or they can initiate a service connection where one node sends messages at a determinate frequency to the requesting

component.

1.2 Contribution

The solution proposed in this paper is a single component that can convert messages from the JAUS standard into the format desired by ROS and that can convert messages from the format expected in ROS to that expected in JAUS. This component will be referred to as a bridge. However, instead of developing a cumbersome component that can handle any data transformation between the two standards, this paper proposes using a code generator to construct only bare bones bridges as necessary. A bare bones approach to these bridges will allow them to remain simple enough to run on he military specification embedded hardware in the CATVehicle without overburdening it. These bridges will also be generated in such a way as to be integrated in future work with a larger project encompassing generating software for the whole of a system developed jointly in ROS and JAUS.

1.3 Testing

Prior to integration with ROS, the CATVehicle has been simulated in a limited JAUS simulator. For this project a car simulation was developed in the ROS based simulator Gazebo [2] to more accurately describe the movements of the physical vehicle and the environment in which is moves. The simulator was developed in such a way as to mimic the commands and procedures necessary to operate the physical vehicle. As such, the simulator and vehicle are interchangeable in so far as the software components that drive the vehicle are concerned.

2. BACKGROUND

The work developed in this paper involves applying aspects of Model Based Design (MBD) to the autonomous vehicles development process. Though it is not discussed in this paper it is the authors' hope that using code generation to construct the ROS to JAUS bridges described herein will reduce the development time for complex autonomous systems and will increase code reusability by introducing ROS nodes into predominantly JAUS based systems.

2.1 Component based systems

Component-based systems are discussed briefly above in Section 1. However in more detail, the purpose of a component-based system is to take a complex problem, break it into smaller portions or tasks to be accomplished, and distribute those tasks as individual programs or components among a network of computers. The system can then take advantage of multiple computers to process information in a parallel manner while communicating along whatever network protocol is desired by the system designers. In the case of this paper, a standard TCP/IP wifi and ethernet network is used to integrate several computers in the CATVehicle and its simulation environment. Component-based frameworks often also have a layer of abstraction on top of the network on which they are running. This abstraction allows for messages to be passed between components of the system regardless of the actual physical layer implementation of the network (as long as the middleware layer provided by the component-based system is correctly implemented).

2.2 Related Work

Some implementations of ROS to JAUS have already been completed. For instance, the Army Research Laboratory (ARL) developed a ROS to JAUS bridge [5]. However unlike the implementation described in this paper, the bridge developed by the ARL does not use code generation to build customized ROS to JAUS bridges.

Figure 1: The CATVehicle

Instead it relies on human developers to adapt existing code to suit their own needs. Also, a team from Case Western Reserve University (CWRU) at the 2010 Intelligent Ground Vehicle Competition (IGVC) used both ROS and JAUS components in their vehicle's software [3]. However, similarly to the ARL project, the CWRU team did not implement a code generator to build their interface between the two standards.

2.3 Previous Work

The work described in this paper on generating ROS to JAUS bridges is meant to be integrated into a larger modeling language currently in development [8, 10]. Called JausML, this modeling language is currently capable of generating everything necessary to build a JAUS system, but does not include anything for generating ROS artifacts from the models. The ROS to JAUS bridge code generator will open the doorway to including those components in future work of JausML.

2.3.1 Skeleton Design Method

JausML is built using the skeleton design method (an extension of the more common template design method) [9]. Building the code generator for the ROS to JAUS bridges to be compatible with the skeleton design method will make integrating the final results with JausML a trivial matter.

2.4 CATVehicle

The testing platform for this project is a modified Ford Hybrid Escape named the Cognitive and Autonomous Testing Vehicle (CATVehicle). As previously stated, the test platform uses JAUS as its backbone. This means that all communications with the computers that manage the low level actuation and control of the vehicle require the use of JAUS. As such, it would be impossible to fully integrate the CATVehicle with ROS. Instead, the joint approach described herein must be used in order to allow the vehicle to operate with open source ROS component packages. The CATVehicle has several locations in the back section of the vehicle where computers can be mounted to interact with the vehicle's internal computers. These computers are hardwired into the vehicle's network with ethernet cables. Fig. 1 shows what the testing platform looks like.

Currently, the CATVehicle is simulated with in a JAUS component with simple bicycle vehicle dynamics [1]. For most test cases where it is not necessary to move the vehicle over long distances

these dynamics approach a reasonable approximation of the actual behavior of the vehicle. However, the maintain a high degree of accuracy another approach is necessary. For this reason, the Gazebo simulator in ROS is desirable since it offers an open environment in which to design an accurate model of the CATVehicle and additional parameters to specify the conditions under which the vehicle is driving (e.g. friction between the tires and the road or the grade of the road). Additionally, the Gazebo simulator developed for the CATVehicle uses a generic algorithm to adapt its parameters to the given scenario so that it performs more accurately than other simulators.

3. METHODS

First, the assumption that was made in building the code generator for this project is that it will always be known exactly what messages will pass through the node. To this effect, it will always be known what data those messages contain and how those messages are being passed along (i.e. as a single message or as a service connection). This is information that will be available at the time that the bridge is being generated. The assumption allows the code generator to handle establishing the connections that ROS and JAUS will need prior to any messages actually being passed. This assumption can be made since the code generator will eventually be integrated into a modeling language that contains all of the relevant information about how, when, why, and where messages will be passed. Additionally, it will be assumed that there exists a one to one correspondence between ROS topics and JAUS messages. That is, if a ROS topic exists, then there is one and only one JAUS message that exists and contains the same data. This assumption is fairly trivial since both ROS topics and JAUS messages are data structures.

For message passing, ROS operates entirely uses service connections (i.e. it uses the publish/subscribe concept to send topics). However, JAUS has the ability to send just a single message without any of the hassle required in setting up a service connection. This is advantages in JAUS, but it creates a bit of an issue when integrating JAUS with ROS. There has to be some sort of reconciliation between sending a single message and a service connection. However, since the bridge can assume that it knows everything about the messages that will be passing through it, those service connections can be established beforehand and used to pass along the single messages that JAUS can send. Fig. 2 shows what the process of setting up these connections looks like. Upon initialization the ROS node has to communicate with the bridge to be able to receive the information that the JAUS component will eventually send. This assumes that the bridge has already stated to the ROS portion of the network that it is capable of publishing the message that it will receive from the JAUS component.

Fig. 3 shows the process that occurs when a ROS node needs to establish a service connection to a JAUS component. Fig. 3 assumes that the JAUS component is already set up to allow other components to establish a service connection with it and that the bridge has similarly already told the ROS portion of the system that it can publish those messages. Also, after establishing the service connection, the JAUS component would continue to repeat the block that writes the service connection messages. The reverse of this scenario is also possible in a similar manner. A JAUS Component could establish a service connection with the bridge which would then subscribe to content that exists on a ROS node. Assuming of course, that the bridge has already set itself up to establish service connections of those type and that the ROS node is set up to publish

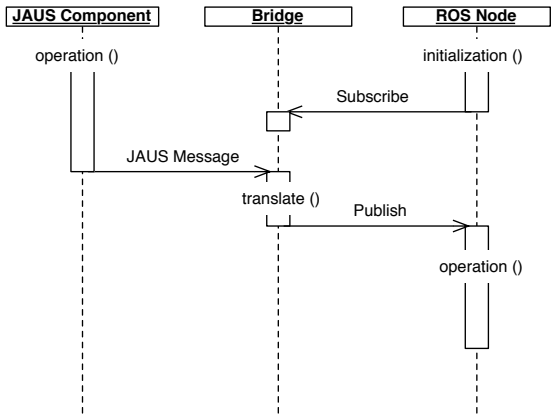

Figure 2: A single message can be sent from a JAUS component to a ROS component even though ROS has to subscribe to the content of the message beforehand.

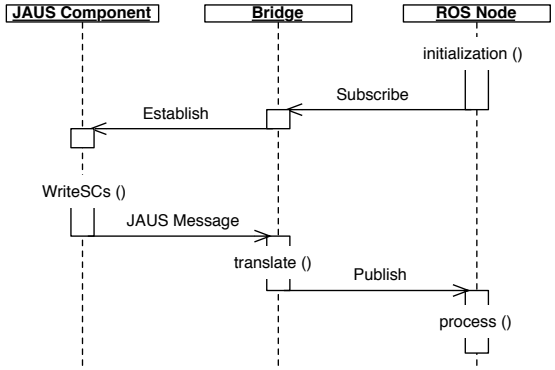

Figure 3: Service connections can be established by establishing them at the bridge and publishing from the bridge.

that information.

As an additional note: JAUS service connections operate at a given frequency (declared during the initialization process). If data does not appear on the receiving end at that desired frequency, then that service connection may become inactive, causing problems with data transmission. The ROS bridges described herein make certain that service connections on the JAUS side of the system stay active as long as the corresponding JAUS components and ROS nodes are alive.

3.1 Modeling the Bridge

Using the modeling syntax from JausML, Fig. 4 shows a single bridge serving as a bridge between many different JAUS components and ROS nodes. Technically, a bridge could be generated and deployed any time a connection between a ROS node and a JAUS Component is necessary. However, it will likely prove more efficient to collect the different types of connections that will need to be made into just one bridge or possibly just a handful of bridges, depending on the requirements of the system. To this effect, the code generator for building these bridges is capable of being building bridges with multiple connections. Also, it should be noted that the bridge can also manage multiple JAUS components subscribing

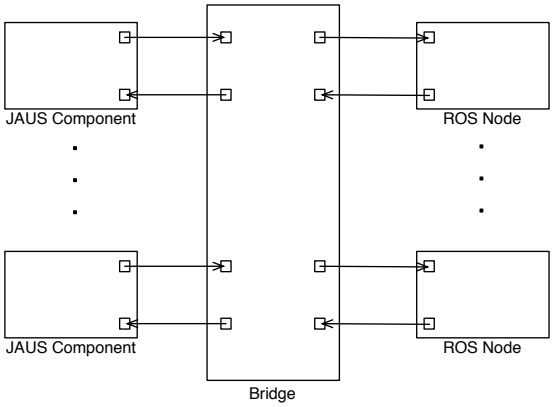

Figure 4: One bridge can service many components.

to the same ROS data and similarly multiple ROS nodes establishing the same service connection. This cuts down on network traffic by preventing the publishing node/component from having to send multiple messages to the bridge to serve the same data to multiple nodes/components on the other side of the bridge.

In a modeling environment it will likely be the case that either the user will be able to determine how many bridges the system should have and exactly what connections each bridge should manage or this process will be managed by a heuristic. However, it should be made clear to the reader that the code generator has not yet been integrated into the larger JausML project. For now it is a stand alone product that can generate bridges with multiple connections.

4. RESULTS

Prior to the development of the ROS to JAUS bridges a suite of examples have been done illustrating different control aspects on the CATVehicle both in simulation and in the vehicle proper. One of these was chosen to demonstrate the ROS to JAUS bridge for this paper. The vehicle has a control algorithm designed to follow a preplanned trajectory with dead reckoning and another control algorithm to limit the speed of the vehicle during a turn so that it remains safe and does not flip or slide (while driving on a flat, non-slippery surface). The steering and velocity controllers were then used to drive the vehicle along a right turn. Success in these results is indicated by a path that appears to be a right turn and a ratio between tire angle and velocity that does not exceed a predetermined value by the velocity controller. All of the paths shown for the vehicle were either recorded directly from the simulated vehicle's state or were recorded using a GPS/INS system mounted inside of the physical vehicle.

4.1 Only JAUS

The original control software for the CATVehicle has been thoroughly tested in the field and in simulation. The results of running a right turn with this software in simulation and in the physical vehicle can be seen in Fig. 5. Also, Fig. 6 shows a plot of the velocity of the vehicle versus the tire angle of the vehicle to show that the vehicle safely and quickly moves through the turn. Due to dead reckoning the vehicle does not follow the path with complete accuracy, but it is obvious that it is able to follow the path none-the-less.

4.2 Gazebo Simulation

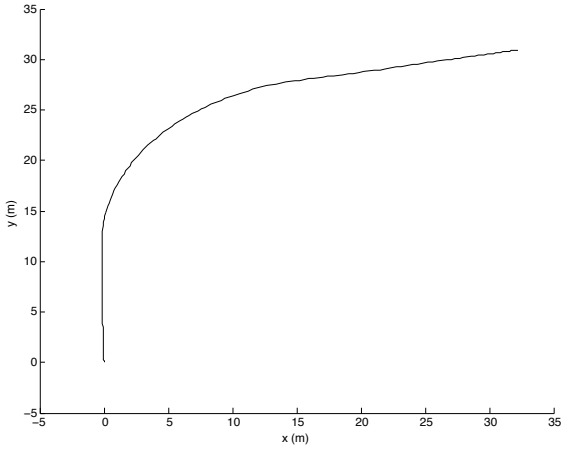

Figure 5: The actual vehiclesáttempts at following the prescribed right turn path with JAUS only.

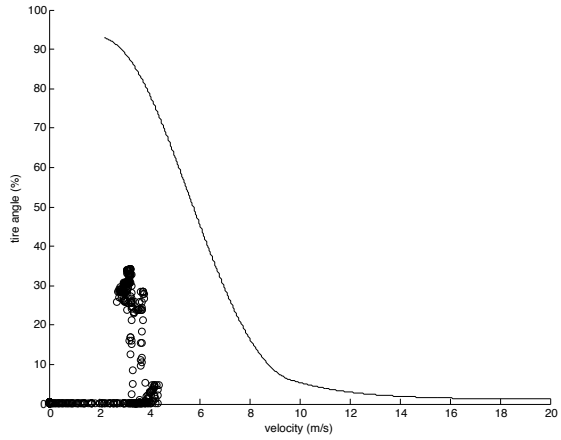

Figure 6: Velocity plotted against tire angle for the JAUS only right turn. The solid line indicates the division between safe and unsafe maneuvers.

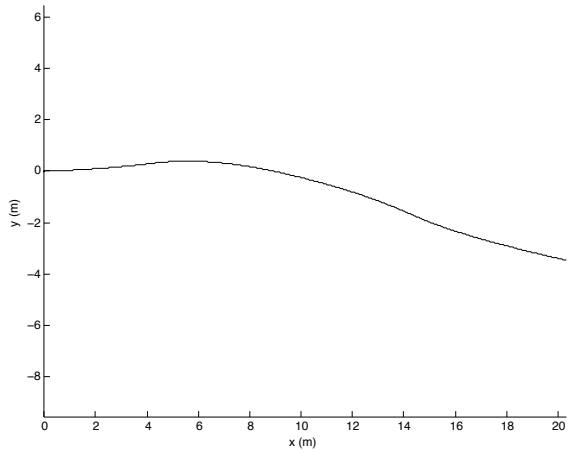

Figure 7: The simulated vehicleś attempts at following the prescribed right turn path in Gazebo.

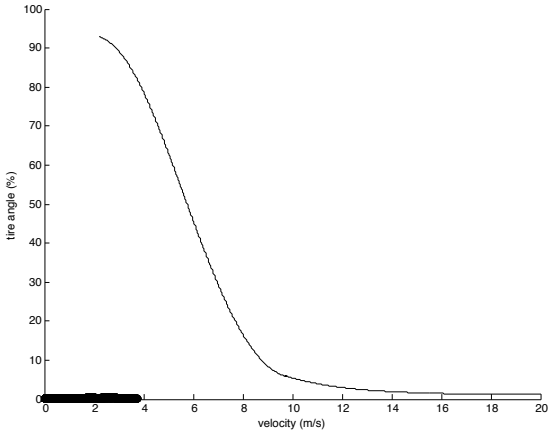

Figure 8: Velocity plotted against tire angle for the JAUS only right turn. The solid line indicates the division between safe and unsafe maneuvers.

Two scenarios were developed for the purposes of testing the bridge with the right turn. The first test involves the Gazebo simulation that was developed for the CATVehicle. A bridge was generated to communicate between the existing control algorithms in JAUS and the new simulator in ROS. The simulated vehicle then executed the right turn while recording its actual trajectory. Fig. 7 shows the results of driving this turn and according to the trajectory. Unfortunately, the Gazebo simulator is an intensive application that requires more computing power to run a successful right turn. However, it can still be seen from Fig. 7 and Fig. 8 that the ROS to JAUS bridge is functioning as intended. The problem exhibited here is that the Gazebo simulator was only able to achieve a 0.25 ratio between real time and simulation time (i.e. 4 real seconds passed for every 1 second of simulation time). The result is that the controller which was running dead reckoning and not actually able to sense the current position of the vehicle was highly inaccurate in prediction the simulated vehicle's actual position. Also, the vehicle in simulation starts at the origin and drives to the right on the graph while the physical vehicle starts and the origin and drives in the positive y direction first. This is simply due to the different starting positions of the vehicle.

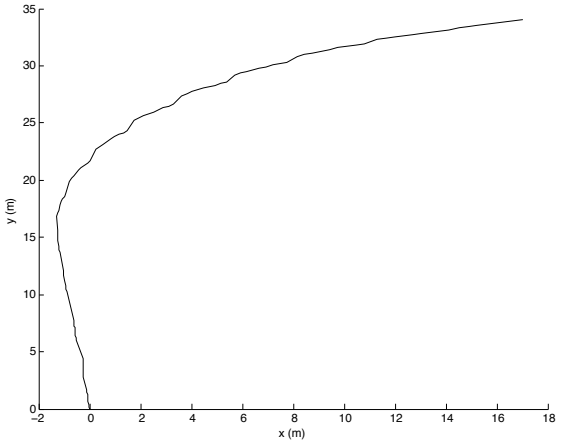

Figure 9: The simulated vehicleś attempts at following the prescribed right turn path in Gazebo.

4.3 ROS Controlled CATVehicle

The second involves porting the control algorithms for the vehicle to a ROS node. A bridge is then set up to allow for communication between the physical vehicle and the ROS node running the control algorithm. This software is then deployed to the physical CATVehicle for operation. Fig. 9 shows the path taken by the vehicle as recorded by the GPS/INS system while Fig. 10 shows the tire angle versus velocity for the ROS driven vehicle. In this test some accuracy issues can be seen as the GPS/INS system locks onto the car. More noticeably from Fig. 10, the ROS driven vehicle is reacting more wildly to the turn than the JAUS only vehicle (the reader will note from the figure that it still remains within the safe operating region). This is likely due to the additional delay caused by the bridge. It is likely that further improvements to the performance of the bridge would fix this issue. However, the vehicle is still capable of safely following the given trajectory. Also, it is possible that some of the issue may lay with the ROS implementation of the control algorithms and not with the bridge.

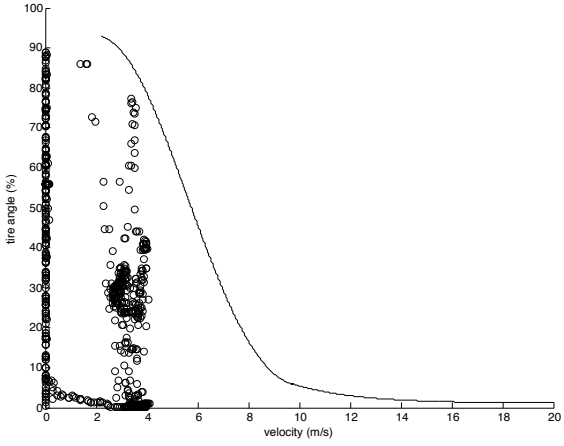

Figure 10: Velocity plotted against tire angle for the JAUS only right turn. The solid line indicates the division between safe and unsafe maneuvers.

5. FUTURE WORK AND CONCLUSION

As illustrated by the examples above, there are some improvements that can be made in the performance of the generated JAUS to ROS bridge. Most importantly for the discussion of this paper, the examples show that the bridge needs to be capable of more quickly passing messages along. The first step in this process will be determining how much of a bottleneck the bridge actually is for the hybrid ROS/JAUS system. The second step will be to take the necessary steps to cut down the latency between when a message is sent on one side of the system and when it is received on the other. It is likely that the issues seen in the results of these generated bridges are due to performance issues with the ROS or JAUS code rather than the interaction of the two.

As previously mentioned, the code generator described in this paper is going to be integrated into the larger JausML project in order to incorporate open source ROS products in the CATVehicle. The integration process should be fairly straight forward since both the bridge code generator and JausML use the skeleton design method.

Additionally, more work is needed to fully integrate the Gazebo simulator with the CATVehicle project through the ROS to JAUS bridge. There are two possible routes that can be explored in fixing the issues in communicating in real time with Gazebo. First, it should be possible to limit the rate at which the JAUS components send information to the simulator based on the ratio between real time and simulation time. However, this step would require rebuilding the logic behind the JAUS components and the rate at which they process data and send new messages. Second, the Gazebo simulator could be ported to more appropriate hardware or simplified to improve performance. However, this second scenario takes the performance of the simulator out of the developersʹhands which is undesirable.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] K. J. Åström, R. E. Klein, and A. Lennartsson. Bicycle dynamics and control. *IEEE Control Systems Magazine*, 25(4):26–47, August 2005.

[2] D. Coleman. Gazebo ROS API, June 2013.

[3] B. B. K. L. C. R. M. K. David Thorndike, Eric Perko. Harlie. Project description, Case Western Reserve University, 2010.

[4] Y. Kanayama, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. In *1990 IEEE International Conference on Robotics and Automation*, pages 384–389, May 1990.

[5] J. R. L. Sadler, C. Rao and H. Nguyen. *ROStoJAUSBridge Manual*. Army Research Laboratory, March 2012.

[6] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[7] SAE AS-4:2010. *JAUS Standard*. SAE International, Warrendale, PA, June 2010.

[8] S. Whitsitt and J. Sprinkle. Message modeling for the Joint Architecture for Unmanned Systems (JAUS). In *Proceedings of the 8th IEEE Workshop on Model-Based Development for Computer-Based Systems*, pages 251–259, April 2011.

[9] S. Whitsitt and J. Sprinkle. Model based development with the skeleton design method. In *20th IEEE International Conference and Workshops on the Engineering of Computer Based Systems*, page (in press), 2013.

[10] S. Whitsitt and J. Sprinkle. Modeling autonomous systems. *AIAA Journal of Aerospace Information Systems*, pages (in press, accepted in final form), 2013.