2019 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
AUTONOMOUS GROUND SYSTEMS TECHNICAL SESSION
AUGUST 13-15, 2019 - NOVI, MICHIGAN

INTEGRATION OF THE AUTONOMOUS MOBILITY APPLIQUÉ SYSTEM INTO THE ROBOTIC TECHNOLOGY KERNEL

David Pirozzo[1], Joshua P. Hecker, PhD[2], Alan Dickinson[2], Tim Schulteis[2], Jeff Ratowski[1], Bernard Theisen[1]

[1]U.S. Army Combat Capabilities Development Command Ground Vehicle Systems Center, Warren, MI
[2]Lockheed Martin Corporation, Littleton, CO

ABSTRACT

U.S. Army Combat Capabilities Development Command (CCDC) Ground Vehicle Systems Center (GVSC) has been managing and developing a variety of autonomous systems throughout its existence. Two of the most important from the past decade include the Autonomous Mobility Appliqué System (AMAS) developed by Lockheed Martin Corporation (LMC) and the Robotic Technology Kernel (RTK) developed by GVSC in collaboration with DCS Corp and Southwest Research Institute (SwRI). Rather than continuing to develop and maintain two separate autonomous software systems, GVSC has decided to integrate any capabilities that were unique to AMAS into RTK and devote efforts to developing RTK going forward. The goal of integrating AMAS into RTK is to leverage the best features of each system. The process of this integration involves multiple steps. This paper describes the historical and current efforts involved in the integration of AMAS into RTK.

Citation: D. Pirozzo, J.P. Hecker, A. Dickinson, T. Schulteis, J. Ratowski, and B. Theisen, "Integration of the Autonomous Mobility Appliqué System into the Robotic Technology Kernel", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 13-15, 2019.

1. INTRODUCTION

The U.S. Army Combat Capabilities Development Command (CCDC) Ground Vehicle Systems Center (GVSC) has been managing and developing a variety of autonomous systems throughout its existence. Two of the most important from the past decade include the Autonomous Mobility Appliqué System (AMAS) developed by Lockheed Martin Corporation (LMC) and the Robotic Technology Kernel (RTK) developed by GVSC in collaboration with DCS Corp and Southwest Research Institute (SwRI). Rather than continuing to develop and maintain two separate autonomous software systems, GVSC has decided to integrate any capabilities that were unique to AMAS into RTK and devote efforts to developing RTK going forward.

AMAS has focused on developing autonomous driving capabilities for logistics vehicles. One aspect of this is to modernize military vehicles by providing driver warning and assist features found on most consumer vehicles such as blind spot detection, cruise control, lane keeping, and collision mitigation braking. In addition to these driver assist features, AMAS provides autonomous driving capabilities such as teleoperation, waypoint following, and convoy vehicle following. The software and hardware have undergone thorough testing, giving AMAS a high level of reliability.

RTK has been focused on developing new perception and autonomous navigation capabilities for smaller, more mobile vehicles. The software is built upon the open source Robot Operating System (ROS) which provides simple communication mechanisms between software components and gives developers tools for monitoring and diagnosing the state of the system. ROS also promotes modular design allowing for more flexibility and reusability of individual software components. RTK has demonstrated this by reusing software across many different vehicles, supporting many different programs with varying objectives, and a quick turnaround time to integrate new capabilities.

The goal of integrating AMAS into RTK is to leverage the best features of each system, creating a library of compatible capabilities that can be pulled from to support future autonomous systems programs such as Next Generation Combat Vehicle (NGCV) and Combat Vehicle Robotics (CoVeR). AMAS components that relate to multi-vehicle coordination and leader/follower behaviors help to fill capability gaps of RTK since it has always been used on single vehicle systems. A long-term goal of RTK is to help define an open architecture for autonomous ground systems. By integrating new components into RTK, the interface definitions are re-evaluated and redefined, making them more robust and allowing the architecture to accommodate a wider array of components.

2. INTEGRATION PROCESS

The integration of AMAS into RTK has been an ongoing effort for several years. The scope of the effort began with a small team trying to perform a loose integration and has grown to a larger team trying to achieve a fully integrated solution

2.1. Early Efforts / BWASK Integration

Early efforts focused on simply controlling an AMAS By-Wire Active Safety Kit (BWASK) using the RTK Autonomy Kit (Akit). This work primarily focused on developing the set of control and status interfaces between the two systems, without modifying the behavior of either system. Within the AMAS Akit, a set of Neutral Message Language (NML) message buffers were used in a polling model to share data between modules, similar to how ROS topics are used for communication between nodes. The Autonomy Gateway module within the AMAS Akit performs serialization of these NML messages and uses a combination of Ethernet and Controller Area Network (CAN) interfaces to handle sending and receiving data between the Akit and BWASK. Rather than recreating the code for managing these low level interfaces, the Autonomy Gateway and NML buffer modules were utilized by RTK and a series of NML wrapper classes were created which converted data between ROS topics and NML buffers. Some of the high-level support classes for interfacing with the NML buffers were taken from AMAS to simplify the development of these wrapper nodes.

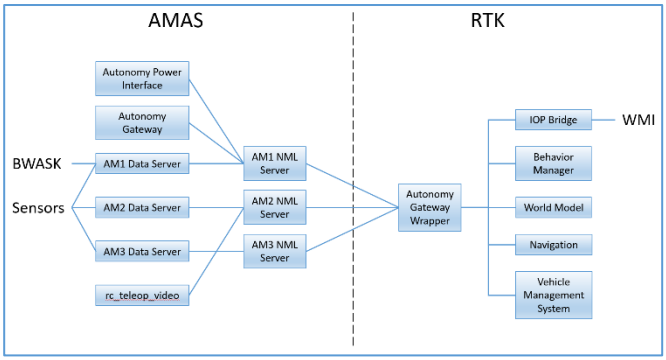

Figure 1: Initial approach to integration using Autonomy Gateway Wrapper node.

Once the wrapper nodes were created, this provided the means for passing data between the RTK Akit and the BWASK; however, there were still a number of issues preventing the two systems from cooperating with one another. For example, even though the path following commands were being sent to the BWASK using the correct message format, the way that RTK generates the shape of its paths did not meet the strict list of requirements that the BWASK path executor expected, resulting in paths often being treated as invalid data. Other issues occurred within the RTK Vehicle Management System (VMS), which performs checks on the system's health and status data to ensure that requirements are met when executing autonomous behaviors. To account for the variations in available status information, new configurations had to be added to the VMS to allow for different or missing status information. Lastly, the BWASK provides a long list of Built-In-Tests (BITs) that continuously monitor the system for errors, similar to the VMS. When something is wrong, these BITs will trigger persistent errors that require a full reset of the system to be cleared. Unfortunately, many of these BITs seemed to be intermittent and were not issues that were able to be resolved, making them an inconvenience that caused frequent interruption to testing and development.

Once the various inconsistencies between the RTK and AMAS systems were resolved, it was possible to successfully control the BWASK using RTK behaviors such as teleoperation and waypoint following. This was the primary goal for the initial phase of integration; however, work continued to further integrate capabilities. The next priority was to utilize Light Detection and Ranging (LIDAR) and Radio Detection and Ranging (RADAR) data available through the BWASK to populate the RTK world model. It was possible to capture the data from the sensors; however, since RTK had minimal experience with processing Ibeo LIDAR or Delphi RADAR data, the results were very noisy and not very useful for doing obstacle detection and obstacle avoidance (ODOA). The last goal was to harness the AMAS Akit modules for performing basic leader/follower behaviors. There was some initial success with performing leader detection; however, the work required to pass that data to the follower and execute the estimated trajectories was never completed. Shortly after this work began, the team transitioned their efforts to supporting the Coalition Assured Autonomous Resupply program and took a different approach to Akit integration.

2.2. Later Efforts / Initial Akit Integration

As the CAAR program started, the integration team grew to include support from Lockheed Martin, allowing for tighter integration between AMAS and RTK. This effort shifted the focus from controlling only the BWASK and making use of its sensor data, to integrating AMAS Akit capabilities, in particular those features which would allow RTK to perform leader/follower behaviors. This effort was still not focused on changing any of the behaviors, but rather on developing the interfaces between RTK Akit components and AMAS Akit components so that they could coexist within one system.

To demonstrate tighter integration, RTK and AMAS were compared side by side and modules that provided duplicate capabilities were identified as places to focus efforts. The overall goal was to integrate novel capabilities from AMAS into RTK,

with a priority of using RTK modules whenever possible and using AMAS components to fill specific capability gaps. After performing the comparison, it was determined that RTK components would be used for behavior management, world modeling, and system health monitoring, while AMAS components would be used for hardware interfaces, inter-vehicle communication, leader detection and tracking, convoy trajectory generation, and convoy gap control. The interfaces between the two systems are shown in Figure 2 below.

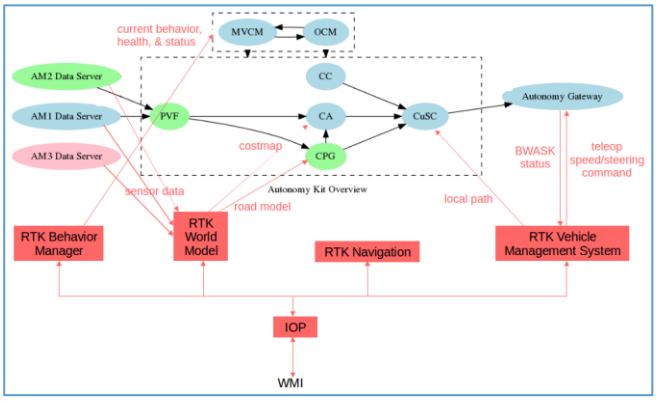

Figure 2: Interfaces between AMAS (top) and RTK (bottom) modules for convoy behaviors

It was decided that the quickest, easiest, and lowest risk way to allow RTK and AMAS to communicate was to continue the approach of developing wrapper classes that would convert messages between NML and ROS, similar to what was done for the Autonomy Gateway, but applied throughout the entire system. These wrappers would perform any necessary translation between RTK and AMAS message types, and minimal changes would be required to the AMAS autonomy modules. This approach was a much larger effort than expected, and the resulting code was overly complex, making it very difficult to work with.

Seeing the results of this effort, it was clear that the use of wrappers would not support the desired level of integration between RTK and AMAS, and further redesign was required.

2.3. Current Efforts / Full Akit Integration

The current approach to integrating AMAS into RTK involved heavily refactoring the AMAS software to better fit within the RTK system design. The main tasks required converting NML interfaces to ROS, decomposing subsystems into modular components, and updating interfaces to adopt RTK standard message formats.

2.3.1 ROS Conversion

The full integration of the AMAS codebase into the RTK framework began with a module-by-module conversion of the AMAS build system. The migration from SCons, AMAS's previous Python-based build system, to catkin, ROS's official CMake-based build system, was a relatively straightforward conversion process with few exceptions. For example, several Python build scripts had been extended to perform complex tasks, such as code generation or compile-time decision making, that could not be easily ported to CMake and had to be carefully refactored out. Each AMAS submodule was restructured and *catkinized* following ROS style guidelines and naming conventions, and code repositories were forked to mitigate possible conflicts with other development projects.

In the second stage of conversion, custom ROS messages were created to replace every NML message class identified as a part of the AMAS codebase. Because NML messages are defined by C++ classes, they typically contain data, in the form of variables, and methods, which can be applied to this data; ROS messages, on the other hand, may only contain data. Therefore, care was taken to ensure a one-to-one translation of NML message data to ROS message data whenever possible to simplify the subsequent message replacement process within the source code itself. If necessary, the NML message methods were retained in the form of generic helper utilities that could later be

applied to the ROS messages to replicate the previous NML functionality.

For the third stage, ROS publishers and subscribers were inserted into the AMAS codebase in parallel with existing method calls that read from, or wrote to, the NML shared memory buffers, using the newly defined ROS messages as payload. Note that the NML buffer calls were left in place as a form of *scaffolding* to support testing during the ROS transition period, ensuring inter-communication between modules whether or not they had been converted to ROS. A major challenge during this stage was adapting the transitory nature of ROS messaging to support the storage-like functionality of NML buffers – several key functions of the AMAS codebase were written with the expectation of on-demand data availability and automatic data staleness tracking, therefore ROS's event-driven callbacks had to be restructured to support this need.

In the final stage of ROS conversion, preprocessor directives were inserted throughout the AMAS codebase to wrap all NML-dependent blocks of code, giving the developer the ability to remove all NML-related code at compile time by setting a CMake compiler flag to ***TRUE***. Following the standard set throughout the rest of the conversion process, this compile-time mechanism provided a fine degree of control over which portions of the codebase were exclusively using ROS and ensured a maximum amount of reversibility for identifying errors or bugs inadvertently introduced during the conversion.

2.3.2 Refactoring AMAS

After updating the AMAS codebase to be ROS-compatible, major remaining conversion tasks included the modification of program control flow (switching from procedural to event-driven), the standardization of vehicle-specific parameters (using ROS's shared network parameter server), and the unification of vehicle coordinate transforms into a single ROS *tf* tree structure (using Unified Robot Description Format (URDF) files). As with the previous effort, conversion was performed on a per-module basis to ensure that each module could be tested independently, while the integrity of the full AMAS codebase was maintained to ensure that the entire system could be evaluated at any time.

The main challenge of this refactoring process was deciding how to best preserve the high-level autonomy behaviors of AMAS while simultaneously migrating toward an RTK-style flow of control. The primary goal was to ensure that the overall periodicity of the refactored system was driven by the rate of the incoming sensor data. Approximate synchronization was achieved using ROS's *message_filters* library, which unites all sensors required for a given processing decision into a single event-driven callback method, then triggers the method once all required sensor data has arrived. Additional enhancements included the decomposition of monolithic, end-to-end AMAS services into modular components that better match RTK's functional layout, the removal of the custom-built, multi-threaded AMAS data servers (replaced by a handful of memory-sharing ROS nodelets), and the creation of per-node launch files to support standalone running and testing of individual components.

2.3.3 RTK Integration

The team began by identifying core functionality and the modules in which that individual functionality resided in AMAS that needed to be integrated. At the same time the team identified the modules within RTK where the AMAS functionality would need to be integrated. From this effort, the team created a map of the new architecture depicting the grouping, interfaces and interactions of the integrated modules.

Figure 3(A) illustrates the completed integration of AMAS into the RTK framework as it interfaces with the AMAS BWASK. Figure 3(B) illustrates a

variation on this integration with modifications to interface with the IDS drive-by-wire kit.

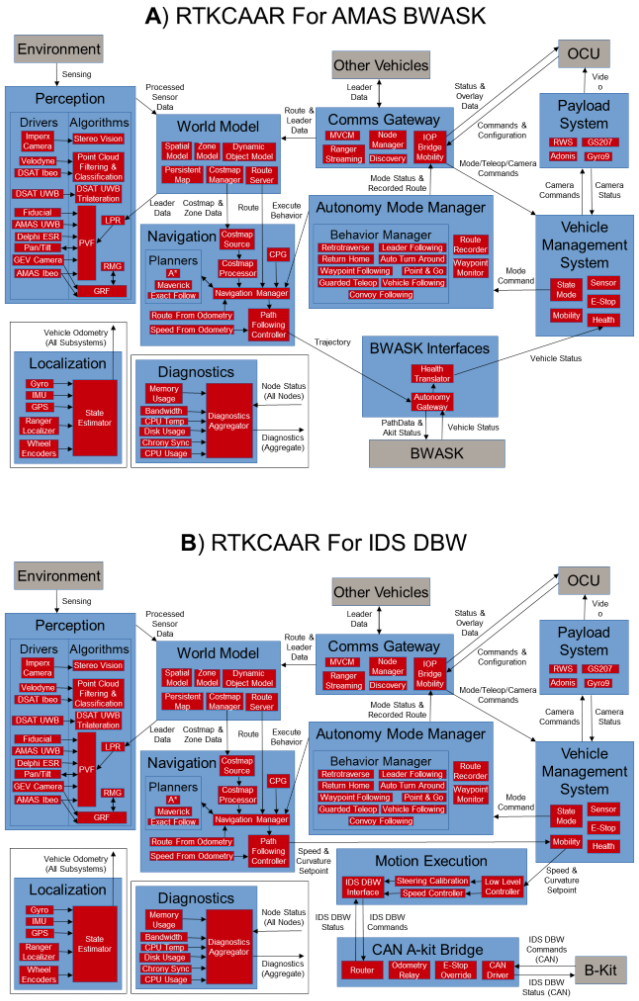

Figure 3: Variations of the RTK architecture with AMAS fully integrated: **A)** RTK operating on-board military transport trucks, interfacing with the AMAS BWASK; **B)** RTK operating on-board a HMMWV, interfacing with the IDS drive-by-wire actuation system.

3. RESULTS

The integration of AMAS into RTK has been a challenging but rewarding effort and has resulted in a more cohesive and modular design than was previously available in the standalone AMAS system. It provides flexibility by utilizing the AMAS convoy capabilities and sensors on other similarly-equipped vehicles to be able to interoperate in heterogenous vehicle convoy resupply scenarios and missions. We will soon be demonstrating a 6 vehicle convoy with two LMTVs, two HX-60, and two HMMWVs working together to perform the capstone defensive formation demonstration at Camp Grayling, MI. The RTK architecture provides a common framework for integrating new capabilities and modes into autonomous ground vehicle systems. RTK itself is built on top of ROS which also provides a very modular and flexible architecture for the development of loosely coupled modules communicating with each other for a common purpose. Integrating the AMAS autonomy kit functionality into ROS, and specifically RTK, provides new multi-vehicle and convoy capabilities that did not exist in RTK prior to this effort. New modes and multi-vehicle data sharing capabilities have been added to support the convoy operations. These can be utilized for other multi-vehicle modes and data sharing applications and capabilities (i.e. obstacle sharing).

There were several challenges that the team had to overcome. First, working in a distributed team with multiple co-contractors and government partners required a method for sharing the software repositories and information. We decided to utilize the Defense Intelligence Information Enterprise (DI2E) framework to provide collaboration capability. This can be difficult to get access to as it is government controlled and requires special credentials, but does provide a stable method of sharing the software once everyone is approved. DI2E is able to host all the Git repositories for both the RTK architecture and AMAS software and provides collaborative tools available for the team to keep everything up to date and track issues and tasks. In the future, we would like to utilize the continuous integration capabilities of DI2E to ensure that the software integrity is maintained on the site.

Second, vehicle hardware-related development and integration efforts can be very challenging. Difficulties can arise with hardware anomalies or software bugs that can be difficult to pinpoint the source and develop a solution for without extensive time and effort to track down. Combined with multiple vehicle types, this adds an additional level of difficulty with possible vehicle-specific differences. RTK and AMAS are looking to minimize this effort by isolating the hardware interfaces into standard ROS and RTK messages so the hardware and vehicle-specific elements can be encapsulated in these interface nodes and minimize the impact on the rest of the architecture. AMAS provides an Autonomy Gateway, Power, and Sensor interface layers to isolate the BWASK-specialized hardware and safety critical software from the RTK autonomy kit functions. RTK/AMAS has also developed ROS launch configurations for vehicles that have limited sensor capabilities or are not equipped with the full AMAS sensor suite. For instance, if a vehicle configuration does not have a road-following camera and Ibeo LIDAR, the RTK architecture will look at generating the standard imagery and 3D point cloud data from other available sensors such as stereo cameras and Velodyne LIDAR. This flexibility will make the architecture more adaptable to a variety of vehicles and missions, especially with the incorporation of the multi-vehicle capability that AMAS provides.

Third, we utilized the ANVEL simulation environment with the RTK/AMAS hybrid architecture to provide a testbed for incorporating AMAS functions into RTK in a realistic runtime environment without requiring physical vehicles and the logistics of running those vehicles in convoy configurations. This implementation has its own challenges, but it does provide a mechanism to evaluate the RTK/AMAS system in a reasonable manner before running it on physical vehicles. The ANVEL simulation has some limitations that make it difficult to do full integration testing before moving to the vehicles, especially in raw sensor data. AMAS utilizes several types of sensors such as cameras, LIDAR, and RADAR. Currently, ANVEL can simulate only a single LIDAR, but most of RTK/AMAS vehicles are equipped with at least three LIDAR (two front-facing and one rear-facing). Additionally, ANVEL does not currently provide a simulator for ultra-wide band antennas (UWBs) or RADAR, and these sensors are utilized by AMAS for range and collision information. Therefore, the utility of the ANVEL simulation environment can be somewhat limited for full system testing. However, it is very useful for running AMAS modules and capabilities in RTK architecture and checking out individual module functions and interfaces.

Despite all these challenges, the team has been successful in bringing all of the AMAS capabilities into the RTK architecture and demonstrating these on multiple different vehicles. These AMAS capabilities will provide the addition of local perception-based road following and vehicle tracking, multi-vehicle coordination and data sharing, and adding an overall comprehensive convoy operations capability to RTK.

Figure 4: HMMWVs, LMTVs and HX60 vehicles used in CAAR demonstration.

4. CONCLUSION

The joint Government and industry integration team has completed much work towards the goal of integrating AMAS into RTK. The team identified core functionality and the modules in which that individual functionality resided in AMAS that needed to be integrated. Simualtaneously, the team

identified the modules within RTK where the AMAS functionality would need to be integrated. From this effort, the team created a map of the new architecture depicting the grouping, interfaces, and interactions of the integrated modules. The team then focused on removing the Neutral Message Language (NML) *scaffolding* used in AMAS, setting up the ROS messaging infrastructure that would take its place to support data passage between modules, and establishing the format and extent of the content of the messaging that had to be passed among the various modules. The team then examined which modules would be directly transferred into RTK almost whole, which modules would need to be carefully integrated piecemeal into existing RTK modules, and which modules could be almost entirely deprecated. For the piecemeal transfer of multiple functions within a given module, the module had to be decomposed into separate components so they could be individually integrated into the appropriate location in the RTK architecture. Other supporting tasks included conversion of AMAS data logging into ROS logging. Once the planning was completed, the team began the actual conversion of the module code.

Integration and test events to date show the team is on the right path. Testing of the modules converted to date resulted in initial functionality.

Next steps include the completion of the development of the remaining modules, integration with the rest of RTK, and full system testing. Once the team completes these tasks and formally merges the code, full functionality will be achieved. The result will be showcased in an RTK architecture demonstration using a six vehicle convoy as part of the CAAR program targeted for later in 2019.

After the culmination of this RTK conversion work at the 2019 demonstration, we plan to look at additional improvements. GVSC plans to look at how to make AMAS modules more functional for typical RTK systems which do not have the full suite of sensor hardware that AMAS currently supports. Additionally, GVSC is in the process of beginning to plan out the integration of other software systems into RTK, such as Autonomous Ground Resupply (AGR).

Lockheed Martin has continued to grow its industry leading Autonomy expertise by executing this RTK development. GVSC plans to continue to grow RTK developer teams by reaching out to and including universities and other research organizations in future developments.

The impact of this conversion effort will help RTK to become the de-facto autonomy library for Unmanned Ground Vehicles (UGVs). It will prove out not only that this type of integration is indeed possible, it will also provide lessons learned for future similar efforts integrating code on other systems with RTK. Other ground vehicle autonomy programs such as the Next Generation Combat Vehicle (NGCV) and Combat Vehicle Robotics (CoVeR) are watching the result of this conversion. Once this work is completed, NGCV, CoVeR, and others will be able to leverage capabilities from both RTK and AMAS.