An Overview of an Autonomy Stack that Relies on the VxWorks Real-Time Operating System

Deepak Charan Logavaseekaran, Rakshith Macha Billava, and Dan Negrut

Department of Electrical and Computer Engineering, University of Wisconsin – Madison

December 8, 2023
Contents

1 Introduction 2

2 Solution Description 2
   2.1 Specifications .......................... 2
   2.2 Virtualizing Time ........................ 2
1 Introduction

The ART unmanned ground vehicle (UGV), in its current state, relies on the Robot Operating System (ROS) [1] for communicating between various nodes. ROS is an open-source framework widely known for its ease of management and its plug-and-play nature for rapid prototyping. Although ROS is highly regarded for its interoperability and modularity, it has support of real-time operations, which is crucial in achieving determinism and producing autonomy solutions that guarantee performance. In the recent October 2023 ROS Conference, the developers have identified this and have assured that real-time operations will be supported in ROS2 in the future. For our current needs, however, we need to explore and evaluate other real-time solutions to make an educated decision on whether we need to pivot from ROS.

VxWorks [2] is an attractive and competitive alternative in this domain. VxWorks is a real-time operating system (RTOS) developed by Wind River Systems. It focuses on determinism and has a performance-driven approach that aligns with our goals. As some of the NASA rovers have already been using VxWorks, this would be a worthwhile alternative for us to consider. The goal of the technical effort reported here was to design a bare-bone proof of concept autonomy stack (A-Stack) to measure its performance, explore its deterministic behavior and the overall fit of VxWorks for our lab’s requirements.

2 Solution Description

2.1 Specifications

For the purpose of evaluation, we will be considering an architecture with just 3 nodes – the IMU node, the GPS node, and the control node. The IMU and the GPS nodes would be responsible for providing the respective sensor data. The control node performs the sensor fusion, evaluates the current state of the bot, and provides the next command. The approach that we will be taking is to make these tasks time-driven instead of the previously event-driven ROS nodes. In this way, each of these nodes will be independent of each other and will be running on their own time periods. The IMU and the GPS tasks can run at a time period of \( t_1 \) and \( t_2 \) intervals respectively. The control task then runs independently at the \( t_3 \) time period. It takes the most updated data from these sensor tasks and performs the computation. The sensor data is passed by the sensor tasks to the control tasks through message queues which is supported by VxWorks. In addition to this, there also must exist a manager task which is responsible for launching all the tasks and managing the resources.

2.2 Virtualizing Time

We would also like to explore the concept of virtualizing time in our implementation. The A-Stack is expected to be functional when deployed in 3 subsystems –

- \( L_0 \) which is the simulation environment wherein the sensor data will be provided by the simulation. The frequency at which the data is now provided tends to be slower than the real sensors.
• L1 which is the real hardware. The sensors are physically connected to the A-Stack. Data is read from these sensors and the command is generated. This command then actuates the motors.

• L2 is also a simulation environment. However, the expectation here is that the simulation data appears at a much higher rate than the real hardware. The hope is to have the A-Stack to be functional even with faster frequencies.

The time period of each of the tasks now varies depending on the environment the A-Stack is in. The time period will be scaled appropriately. However, to guarantee the real-time nature of the tasks, we should ensure that the task is completed within its provided time budget or ‘time capsule’. In other words, even though the time periods of these tasks vary accordingly, the task is completed within its time capsule and will wait till it is invoked at the beginning of the next time period. Another key feature that needs to be implemented is the hard deadline. Any failure to meet the time period or the time capsule restrictions should cause the systems to run a recovery phase so that the system is aware of the failure and acts accordingly. However, for our proof-of-concept implementation, we will not be having the recovery phase but we will be exploring ways to enforce these hard deadlines.

Figure 1: Task Structure of the VxWorks A-Stack
References
