Project: WebAssembly for ROS Packages

Date:	19/04/2023
Student Engineer:	Michael Batchelor
Student ID:	N9991000
Supervisor:	Dr. Tobias Fischer

Version	Date	Author	Changes/Comments
1	19/04/2023	Michael Batchelor	Initial Project Proposal

General Objective

This project will build on top of previous work done in the RoboStack project by extending the build process to include *Web Assembly* as a build target. RoboStack is a cross-platform bundling of Robot Operating System (ROS) which uses the Conda package manager. Conda is a cross-platform package and environment management which can quickly install, run, and update packages and their dependencies (Anaconda Inc., 2017). RoboStack implements a continuous integration/continuous deployment pipeline (CI/CD) which aims to produce consistent, reproducible builds of ROS packages for Linux, Mac, and Windows. The compiled binaries are then packaged and uploaded to an Anaconda repository to be distributed by Conda. The goal of this project is to add *Web Assembly* (WASM) as a compile target to RoboStack to produce WASM binaries of ROS packages. To that end we will investigate how can we modify the existing *RoboStack* framework to cross compile ROS packages to *Web Assembly*.

Key Findings from the Literature

There have been previous attempts to get ROS running on the web. The driving factor for these attempts appears to be large due to the accessibility, and portability of *Web Assembly*. *Web Assembly* is a binary-code format which can be run natively by popular browsers. It aims to offer near native performance while also focussing on safe, portable, and compact code (Haas. A., et al, 2017).

Before the emergence of *Web Assembly, Robots as Web* Services (Osentoski. S., et. al., 2011) investigated using *roslibjs* to achieve this. *Ros On Web* (Allwright. M., 2022) is a more recent example that demonstrates ROS nodes compiled to Web Assembly. The functionality demonstrated by *Ros On Web* is similar to what we hope to achieve with this project, however the project is closed source for now and does not appear to provide a way of cross-compiling additional ROS packages. These examples show that there has been a desire for ROS to run on the web for some time, and that it has been done successfully before, albeit in small scale tests.

The implementation of *Web Assembly* as a compile target in Unity (Trivellato. M., 2018) is also a good example which shows how versatile cross-compiling to *Web Assembly* can be. This is promising in terms of the success of this project.

Emscripten Forge (Beier. T., et al, n.d.) is a project like ours aimed at a different set of Conda packages. It makes use of some of the same build tools used by RoboStack, such as *boa*. *Boa* is a build tool for creating *Conda* packages with better performance than the standard *conda-build* tool (QuantStack & Boa contributors, 2020). *Emscripten Forge* and contains recipes and build scripts for cross-compile C/C++ libraries to *Web Assembly* using *boa.* This gives a good basis to work from as they may have encountered and solved challenges that could be faced during this project.

Stakeholders & Resources

The end users of the project will be individuals or organisations using ROS to develop robotics software. Specifically, those interested in interacting with ROS packages/applications in a browser. There is also a particular interest in developing tools for learning and teaching robotics tools that run in a Web Browser to reduce the barrier of entry when getting started with robotics.

The major stakeholder in this project is Dr. Tobias Fischer. He is the project supervisor and a major contributor to the RoboStack project. He will be informed of progress and consulted when necessary. There are also numerous contributors to RoboStack, who may have a vested interested in the outcome of this project, and how it may impact the existing systems.

The resources required to complete the project are,

- Access to RoboStack and its source code.
- Access to the Vinca tool and its source code.
- Access to ROS packages and their source code.
- A workstation for development/testing. The preferred OS would be Linux, but the project can be completed on any platform due to the cross-platform nature of WASM.
- C/C++ to *Web Assembly* toolchain (e.g., Emscripten).
- Anaconda package repository.
- GIT repository with CI/CD capabilities. This can be either local or remote and is needed to test the generated build scripts.

Project Methodology

The goal of this project is to adapt the existing CI/CD pipeline in RoboStack to cross compile ROS packages to web-assembly binaries. As we are building on top of an existing system, the major design work has already been complete, and so this project will need to work within that framework as much as possible. Much of the effort will focus on exploring the changes required to reach this goal. The broad milestones that will be achieved are,

Milestone 1: Cross-Compile C/C++ source code to Web Assembly.

Before moving on to automating the process, we will need to become familiar with the toolchain used to cross-compile C/C++ to Web Assembly. For this milestone, we will put together a simple "Hello World" type C++ application, cross compile this to Web-Assembly, and run it using a WASM runtime.

Milestone 2: Cross-Compile a ROS package to Web Assembly.

Once familiar with the Web Assembly toolchain, we can then focus on cross compiling a single ROS package in isolation. The package we will use is *rclcpp*. This is a core ROS package with minimal dependencies. A simple test application written in HTML/JavaScript will also be developed that demonstrates the use of the cross-compiled package.

In working towards this milestone, we will gain valuable insight into the requirements and challenges that may be encountered when cross-compiling other ROS packages. We will also be able to explore cross-compilation variables, such as compiler flags, linkage, and compatibility options that may need to be considered later.

Milestone 3: Cross-Compile a ROS package to Web Assembly using boa.

For this milestone will design and develop a recipe and build script for *boa* that will cross-compile the same package from the previous milestone to a Conda package.

Milestone 4: Design template build scripts.

Up to this point, we have been focusing on a single ROS package. This milestone will involve designing generic build scripts for boa. This will remove any elements that are package specific from the work done in previous milestones.

Milestone 5: Integrate the build script template into Vinca.

Once a template build script has been designed and tested, it will be integrated into the Vinca build tool. The generation of build scripts will be verified using the same package used in previous milestones.

Milestone 6: Add WASM compile target to *RoboStack*.

For this milestone, we will add *Web Assembly* as a compile target to *RoboStack*. This will involve adding and updating the necessary config files, as well as modifying the GitHub CI/CD pipeline scripts. Here we will also make use of the updated Vinca tool produced in Milestone 5.

Milestone 7: Adding additional packages.

For this milestone, we will begin adding packages to the *Web Assembly* build. As packages are added to the Web Assembly build, patches may need to be added or modified to fix errors in the build process. The timeframe of this milestone is essentially unbound, and it will be worked on until the end of the project as we will want to add as many ROS packages as possible to the WASM build target by the end of the project.

Deliverables

Deliverable	Description	Date
Progress Report	A progress report delivered midway through the project. This will inform stakeholders of progress made, any challenges, next steps, as well as any changes in scope that differ from what is detailed in this proposal.	31/05/2023
Progress Presentation	A short presentation summarising the details outlined in the Progress Report for the benefit of the stakeholders.	To be announced. Between 10/06/2023 and 24/06/2023
Setup and Usage Guide	This document will detail the setup process, any requirements/dependencies, and the intended usage of the final deliverables.	20/10/2023
Design Document	This document will detail design decisions made during the project. This is to aid future maintainers in understanding the project.	20/10/2023
Modified Vinca build tool	A modified Vinca tool will be delivered that can produce the build scripts necessary for cross-compiling ROS packages to web-assembly.	20/10/2023
Modified RoboStack	A modified version of RoboStack will be delivered that uses the modified Vinca tool to run builds that cross-compile ROS packages to Web Assembly.	20/10/2023
Final Project Report	A progress report delivered at the end of the project. This will inform stakeholders of the final outcomes of the project.	TBD: Estimated for End of Semester 2
Final Project Presentation	A short presentation summarising the details outlined in the Final Project Report for the benefit of the stakeholders.	TBD: Estimated for End of Semester 2

Risks, Requirements & Constraints

Requirements

The key requirements for this project are,

1. Integrate into existing build infrastructure by making use of RoboStack build processes and pipelines.
2. Cross-compile ROS packages to Web Assembly.
3. Package compile *Web Assembly* to *Conda* packages.
4. Deploy *Web Assembly Conda* packages to the RoboStack Anaconda repository.

Risks

There is the possibility of encountering a blocking issue that will prevent the project from being completed. For example, we could discover a fundamental incompatibility between ROS packages and Web Assembly. This is unlikely as there as other projects have been successful doing related work (Allwright. M, n.d.) (ROS WASM Suite, 2023). However, if this is the case we would like to discover this early on so the project scope can be adjusted accordingly. The project plan has been developed with this in mind, with smaller, more easily achievable milestones early on, to uncover any of these potential issues.

In the later stages of the project, there could be the potential of damaging packages in the Anaconda repository. This may interrupt the workflow of *RoboStack* users. They may, for example, need to revert to earlier versions of affected packages or wait for a fix to be published. This results in a time, and potentially monetary cost. These concerns will be mitigated by working in a fork of the *RoboStack* repository and implementing the Web Assembly compile target using a local *Conda* channel.

Constraints

This project is being done within the existing RoboStack project. As a result, we need to minimize impact on the existing build systems, and use the tools and processes already set up if possible. This project also should not damage existing support for other compile targets (e.g., Linux).

The build scripts that are designed for this project should also be compatible with *Boa* so that Conda packages can be produced.

Safety & Ethical Concerns

Deploying packages with security vulnerabilities. It could be worth considering the impacts of continuing to host packages with known security vulnerabilities. ROS packages are generally used in tightly controlled systems, but there are cases where security can be a concern. The harm done by these vulnerabilities could be minimised by making the packages less available, or at least making sure the vulnerabilities are acknowledged somewhere.

Licensing has always been a point of concern with software development. For example, building/deploying packages with incorrect licensing practices, could be considered unethical and perhaps introduce legal issues. Software licensing can be complex, and it is not feasible to vet every package built and published by RoboStack. There can also be some ambiguity in how the licenser vs licensee interpret the many different software licenses used. However, there is research being done into automated tools that can detect these potential violations in software packages (Feng. M, 2019) (Hemel. A, 2011).

Quality & Sustainability

The quality of the project will be determined by two main factors. How well the solution integrates with *RoboStack*, and the number of packages successfully cross-compiled to Web-Assembly. When integrating with *RoboStack* we want to minimise impacts on any existing functionality. A poor-quality outcome may result in existing functionality being crippled or removed to achieve the project goals. The number of packages added to the Web Assembly compile target will give a good indication of the effort required to add and maintain a package for this compile target. Many packages indicate less friction and a higher quality outcome.

The sustainability of the project is linked to the quality of the solution. Once complete, the project will be handed off to another team to be continued. If the solution is of poor quality or hard to maintain then it may get replaced and the work will be redone. This is a waste of resources that should be avoided.

The sustainability of distributing source versus precompiled software can also be considered. Compiling software is quite resource intensive and time consuming. Centralising builds and distributing prebuilt binaries may result in less overall resource usage, and increased efficiency in software development. This not only saves developers time, but also reduces power wasted on recompiling packages locally.

Timeline & Deliverables

No.	Focus	Deliverable	Dependency	Release Date/ Milestone
1	Review of literature.	Literature Review (included in Project Proposal)		19/04/2023
2	Project Proposal	Project Proposal	1	19/04/2023
3	Milestone 1: WASM Hello World			30/04/2023
4	Milestone 2: WASM ROS Package		3	14/05/2023
5	Milestone 3: WASM ROS Package Boa Build		4	21/05/2023
6	Milestone 4: Template Build Scrips		5	28/05/2023
7	Interim Progress Report	Progress Report		31/05/2023
8	Interim Progress Presentation	Progress Presentation	7	To be announced.

				Between 10/06/2023 and 24/06/2023
9	Milestone 5: Vinca Integration		8	06/08
10	Milestone 6: RoboStack integration		9	13/08
11	Milestone 7: Additional Packages		10	20/10
12	Project Delivery Report & Oral Defence	Final Project Report Final Project Presentation Modified Vinca Tool Modified RoboStack	11	TBD: Estimated for End of Semester 2

Management of Project Changes

When managing changes in the scope and execution of the project, it is important to maintain a good record of why and how these changes will be made. To that end, if the scope of the project is to change a change request form will be filled out and must be approved by the stakeholders and supervisor.

Once approved this scope of work document will be updated to reflect the change, with the form attached to the appendix to serve as a record of these amendments. *Appendix: Change of Scope Form* is a template containing the details that need to be provided for a change of scope.

Sign off

	SIGNATURE	DATE
STUDENT ENGINEER		
MICHAEL BATCHELOR	M.B.	19/04/2023

Appendix: Review of Literature

Introduction

In this literature review we are investigating the current state of Web Assembly (WASM) as it relates to robotics and embedded systems. The main points of interest are recent attempts to use WASM in this context, and general advantages/disadvantages of Web Assembly as a compile target. As the focus of this project is compiling ROS packages to Web Assembly, we will also investigate work relating to developing Web Applications using Robot Operating System at the current state of work relating to this.

Key themes

RoboStack

This paper provides a basis for the work that will be done throughout this project. It introduces RoboStack, which tightly couples ROS with the Conda package manager. Some of the key advantages RoboStack brings to ROS is reproducible environments, robust versions management, better compatibility with different platforms and Linux distros, distribution of pre-built binaries, isolation of environments, and simpler creation of packages. RoboStack makes it easy to add new packages to the anaconda repository and can be as simple as adding a single line to a config file (Fischer. T., et al, 2022).

Ros On Web

This project hosts demos of ROS nodes running in a web browser and was achieved by cross-compiling ROS nodes to Web Assembly. The demos are based on example code for the C++ ROS client library. This project demonstrates the possibility of having ROS nodes compiled to Web Assembly. This is done with the hope to enable developers to share their ROS systems securely, without the need to install dependencies locally (Allwright. M., 2022). The three demos show the publishers and subscribers, clients and services, and action server ROS concepts running in the browser.

Robots as Web Services

This paper describes some interesting use cases that emerge when ROS can run in a browser. These use cases seem to be applicable to our project as with ROS2 cross-compiled to web assembly, we will be able to develop ROS nodes that can run in a web-browser.

Some advantages of bringing robotics software to the web, is allowing researchers and users to access robots and equipment via the internet. This would enable a variety of web interfaces for phones and computers. This paper also proposes a remote laboratory which could be created using *rosjs*, which could also be applicable to ROS cross-compiled to Web Assembly. This could give researchers access to a common platform for experimentation, and perhaps shared data as well. It may also give some researchers access to expensive, difficult to obtain equipment they may otherwise have been unable to use (Osentoski. S., et al, 2011).

Many of the benefits of the *rosjs* project are just as applicable to this project as our goal of bringing ROS to the web browser is much the same, just using a more modern technology.

Emscripten Forge

Emscripten forge is a project that implements a build pipeline to compile *Conda* forge packages to Web Assembly (Beier. T., et al, n.d.). It makes use of the same build tools as *RoboStack* making it a good guide for the kinds of changes that will need to be implemented in *RoboStack* to cross-compile to *Web Assembly*.

The point of difference between *RoboStack* and *Emscripten* Forge is Vinca. The recipes and build scripts appear to be managed manually by contributors, whereas *RoboStack* aims to reduce the maintainer burden by automatically generating these files.

ROSWASM Suite

This is a set of libraries that can be used to develop ROS nodes that can be compiled to Web Assembly using *Emscripten*. Using the APIs provided by this project you can write ROS nodes in C++. Once compiled to Web Assembly these nodes are then able to communicate with ROS via a WebSocket opened by *rosbridge_server* (ROS WASM Suite, 2023).

This is the same approach taken by to expose ROS to JavaScript via *roslibjs*. It connects to the WebSocket created by *rosbridge_server* to interface with ROS (ROS.org, 2022).

This is an interesting approach to compiling C++ Nodes to *Web Assembly*; however, this project appears to be aimed at developers creating new Nodes or porting old nodes to *Web Assembly* applications. Our project is more general, in that we want to develop a CI/CD pipeline for cross-compiling existing ROS2 packages to Web Assembly.

Unity on Web Assembly

This blog post details how the Unity team implemented Web Assembly as a compile target for the Unity game engine. This is quite an interesting endeavour and demonstrates the power of cross-compiling to WASM nicely. They have posted a top-down mobile game compiled to Web Assembly that runs in the browser. There are no dependencies to install, it just works. They briefly describe the toolchain based on IL2CPP, *Emscripten* and *Binaryen*, they use to compile from C# to C/C++ and then to Web Assembly (or asm.js) (Trivellato. M., 2018).

Given then complexities involved in building a binary from the unity engine this is promising in terms of our potential success in cross-compiling ROS packages to *Web Assembly*.

Conclusion

There have been a few previous attempts to get ROS running on the web with varying goals and levels of success. The portability of Web Assembly is a factor appears to be a big factor that drives this work. Other software projects, such as the Unity game engine have looked to cross-compiling to *Web Assembly* for similar reasons and have had success doing so. Overall, the prior research indicates that not only is this project feasible, but there is a general desire for ROS to run on the Web.

Appendix: Change of Scope Form

Name	*Name of the person proposing the change*
Date	*Date of the proposed change*
Details	*Description of the proposed change. Mentioning,* • *Any deliverable or milestone impacted.* • *Any changes in dates.* • *Any additional milestones or deliverables.*
Reason	*Give the reason why the change is being proposed.*

Sign Off -

SUPERVISOR	SIGNATURE	DATE

Appendix: Glossary

Boa. A package builder for Conda packages.

CI/CD. A method to frequently deliver apps to customers by introducing automation into the stages of app development.

Compiler. Software that converts a program into a machine-code or lower-level form in which the program can be executed.

Cross-Compiler. A compiler capable of creating executable code for a platform other than the one on which the compiler is running.

Conda. An open-source, cross-platform, language-agnostic package manager and environment management system.

GIT. A distributed version control system that tracks changes in any set of computer files.

Mamba. A fast cross-platform package manager that is compatible with *Conda*.

RoboStack. A bundling of the Robot Operating System (ROS) by Open Robotics for Linux, Mac and Windows using the Conda package manager.

Robot Operating System (ROS). The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications.

Toolchain. A set of tools, supporting libraries and header files that help build a program from source to an executable that can run on a machine.

Web Assembly (WASM). A portable binary-code format and a corresponding text format for executable programs as well as software interfaces for facilitating interactions between such programs and their host environment.

References

QuantStack & Boa contributors. (2020). *Boa Documentation. https://boa-build.readthedocs.io/en/latest/*

Haas. A., Rossberg A., Schuff. D., Titzer. B., Holman. M, Gohman. D., Wagner. L., Zakai. A. & Bastien. J. (2017). *Bringing the web up to speed with WebAssembly*. SIGPLAN Not. 52. https://doi.org/10.1145/3140587.3062363

Anaconda Inc., (2017). *Conda Documentation*. https://docs.conda.io/en/latest/

Beier. T., Renou. M., Vollprecht. W., Mabille. J., Tuloup. K., Paredes. I., Brochart. D., Cross. S., Lamotte. J., Traversaro. S. & Hollands. A. (n.d.) *Emscripten forge*. https://github.com/emscripten-forge/recipes

Hemel. A., Kalleberg. Karl., Vermaas. R. & Dolstra. E. (2011). *Finding software license violations through binary code clone detection*. Proceedings of the 8th Working Conference on Mining Software Repositories, Association for Computing Machinery, pp. 63–72.

QuantStack & Boa contributors. (2020). *Mamba Documentation*. https://mamba.readthedocs.io/en/latest/index.html

Feng. M., *et al*. (2019). *Open-Source License Violations of Binary Software at Large Scale*. IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 564-568

Osentoski. S., Jay. G., Crick. C., Pitzer. B., DuHadway. C. and Jenkins. O. (2011). *Robots as web services: Reproducible experimentation and application development using rosjs*. IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 6078-6083

Fischer. T., Vollprecht. W., Traversaro. S., Yen. S., Herrero. C. & Milford. M. (2022). *A RoboStack Tutorial: Using the Robot Operating System Alongside the Conda and Jupyter Data Science Ecosystems*. IEEE Robotics & Automation Magazine, vol. 29, no. 2, pp. 65-74, June 2022

Allwright. M. (2022). *ROS On Web*. https://rosonweb.io

ROS WASM Suite [Computer Software]. (2023). https://github.com/nilsbore/roswasm_suite.

ROS.org. (2022). *Roslibjs.* http://wiki.ros.org/roslibjs

Johann. T., Dick. M., Kern. E. & Naumann. S. (2011) *Sustainable development, sustainable software, and sustainable software engineering: An integrated approach*. International Symposium on Humanities, Science and Engineering Research, pp. 34-39

Trivellato. M. (2018). *Web Assembly Is here*. https://blog.unity.com/technology/webassembly-is-here