ARL-TN-1134 ● SEP 2022

Streamlined Development Pipeline for MAVericks, ARL's Unmanned Autonomous Vehicle (UAV) Software

by Benjamin Linne

ARL-TN-1134 ● SEP 2022

Streamlined Development Pipeline for MAVericks, ARL's Unmanned Autonomous Vehicle (UAV) Software

Benjamin Linne
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2022	**2. REPORT TYPE** Technical Note	**3. DATES COVERED (From - To)** 1 December 2021–1 August 2022

4. TITLE AND SUBTITLE Streamlined Development Pipeline for MAVericks, ARL’s Unmanned Autonomous Vehicle (UAV) Software	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Benjamin Linne	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLW-TD Aberdeen Proving Ground, MD 21005	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-1134
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Benjamin Linne, 0000-0003-0128-8053

14. ABSTRACT
Robotics is a challenging field that requires the convergence of software and hardware to accomplish desired autonomous missions. Critical to any workflow is the automated building and testing of software before deploying to a production environment. This report discusses the importance and creation of a continuous integration/continuous delivery tool used in the software development process of the US Army Combat Capabilities Development Command Army Research Laboratory’s (ARL’s) unmanned autonomous vehicle software research platform called MAVericks. This tool plays a crucial role in the rapid research and development performed at ARL—including automated build testing for simulation and embedded hardware targets, as well as verifying the desired behaviors in a software-in-the-loop simulation.

15. SUBJECT TERMS
Terminal Effects, MAVericks, continuous integration, UAV, automated, testing, deployment, VOXL

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Benjamin Linne
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UU	26	**19b. TELEPHONE NUMBER (Include area code)**
Unclassified	Unclassified	Unclassified			(410) 278-6219

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction and Background

Continuous integration/continuous delivery (CI/CD) is a commonly used tool in software development to automatically build, test, and deploy code. This tool is critical to improving the speed and efficiency of research while ensuring functionality is not hindered when adding or changing new features. Before CI/CD, the software development process was challenging, and with increasing collaborators modifying the code base, any new development risked breaking existing functionality—such as code no longer building, and autonomous behaviors and fail-safes no longer working as intended.

This report focuses on CI/CD integration for the US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory's (ARL's) MAVericks unmanned autonomous vehicle (UAV) software platform, which is built on the open-source platforms ROS2 and PX4. ROS2 is a set of software libraries and tools for building robot applications, and PX4 is a powerful flight control software for UAVs. Leveraging both platforms, MAVericks is a large collaboration focused on agile flight that works across both simulation and robot platforms. MAVericks is targeted to run on ModalAI's VOXL and RB5 hardware platforms because of the size, weight, and power it offers, along with being a Blue UAS program partner, which means they were funded by the Defense Innovation Unit to comply with Section 848 of the 2020 National Defense Authorization Act regulations.[*]

Collaborators include the United States Military Academy's West Point as part of the Distributed and Collaborative Intelligent Systems and Technology program; University of California, Berkeley, as part of the Scaled and Robust Autonomy program; and the University of Maryland's AI and Autonomy for Multi-Agent Systems program—and the list is always growing. Furthermore, ARL is always seeking to increase the robustness of its algorithms and mature capabilities to transition to other organizations within DEVCOM and the DOD. With many collaborators joining MAVericks, it is important to ensure a minimum useable functionality after every modification to encourage rapid onboarding and contributing. MAVericks is a large research platform composed of over a hundred packages and it is important that every package builds and works reliably. Often, collaborators are only concerned with a few packages, and it is critical that they can easily make changes and additions without needing to troubleshoot unrelated issues. Due to this growing community, it is easy to incidentally introduce bugs or

[*] More information can be found at https://www.congress.gov/bill/116th-congress/senate-bill/1790.

break unrelated functionality. Therefore, CI/CD is an excellent solution that will increase the reliability and usability of the platform for a diverse group of users.

The CI/CD pipeline enables many features that streamline development. It can fully build the entire platform, ensure the dependency installation is successful for new users, run and test the platform in the simulation environment to ensure autonomous behavior is working correctly, and rapidly build compressed workspaces to prevent the need to build on the UAV.

One problematic scenario in software development for autonomous systems is when a user modifies several packages, but only builds and tests one specific package. Thus, code is merged into production without verifying it will work for others. If the untested changes are merged, packages that depend on those changes may no longer build or pass all test cases.

From a user's perspective, CI/CD is triggered by a user creating a code Merge Request (MR) to add their changes to the main branch. This initiates CI/CD with a pipeline that is created. The pipeline comprises four stages: build-dependencies, build-full, test, and deploy. For each stage, multiple jobs can be run in parallel to complete the stage. In each job, the pipeline first copies the merged changes into a fresh environment and completes a specific task. At the end of the pipeline, a fully built version is uploaded and ready to be flashed on a UAV. If any step fails, the remaining pipeline stages are aborted and the user is notified of exactly what went wrong, so they can fix any problems. See Fig. 1 for an overview of the pipeline.

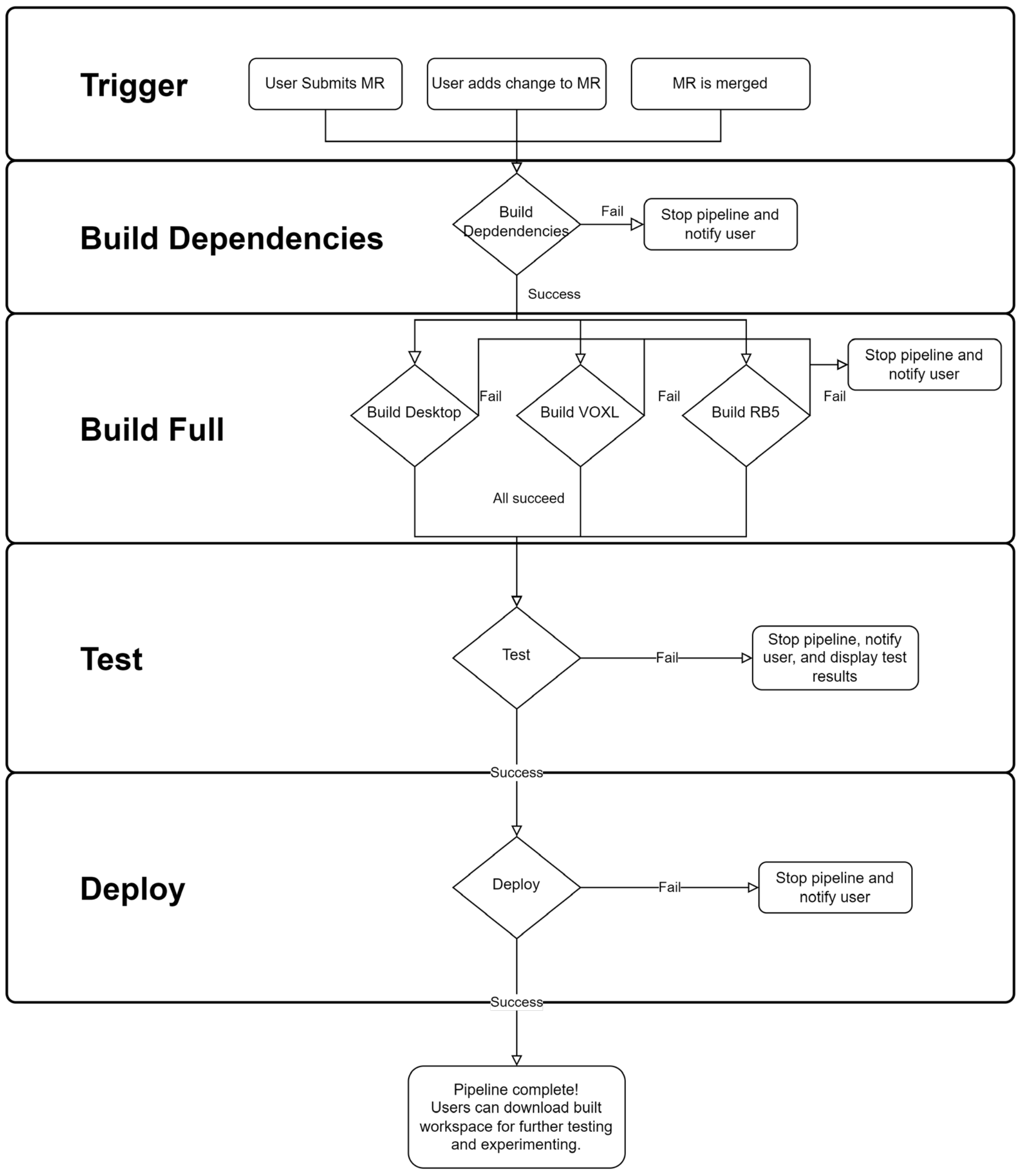

Trigger
User Submits MR
User adds change to MR
MR is merged
Build Dependencies
Build Depdendencies
Fail
Stop pipeline and notify user
Success
Build Full
Build Desktop
Build VOXL
Build RB5
Fail
Fail
Fail
Stop pipeline and notify user
All succeed
Test
Test
Fail
Stop pipeline, notify user, and display test results
Success
Deploy
Deploy
Fail
Stop pipeline and notify user
Success
Pipeline complete! Users can download built workspace for further testing and experimenting.

Fig. 1 Pipeline overview

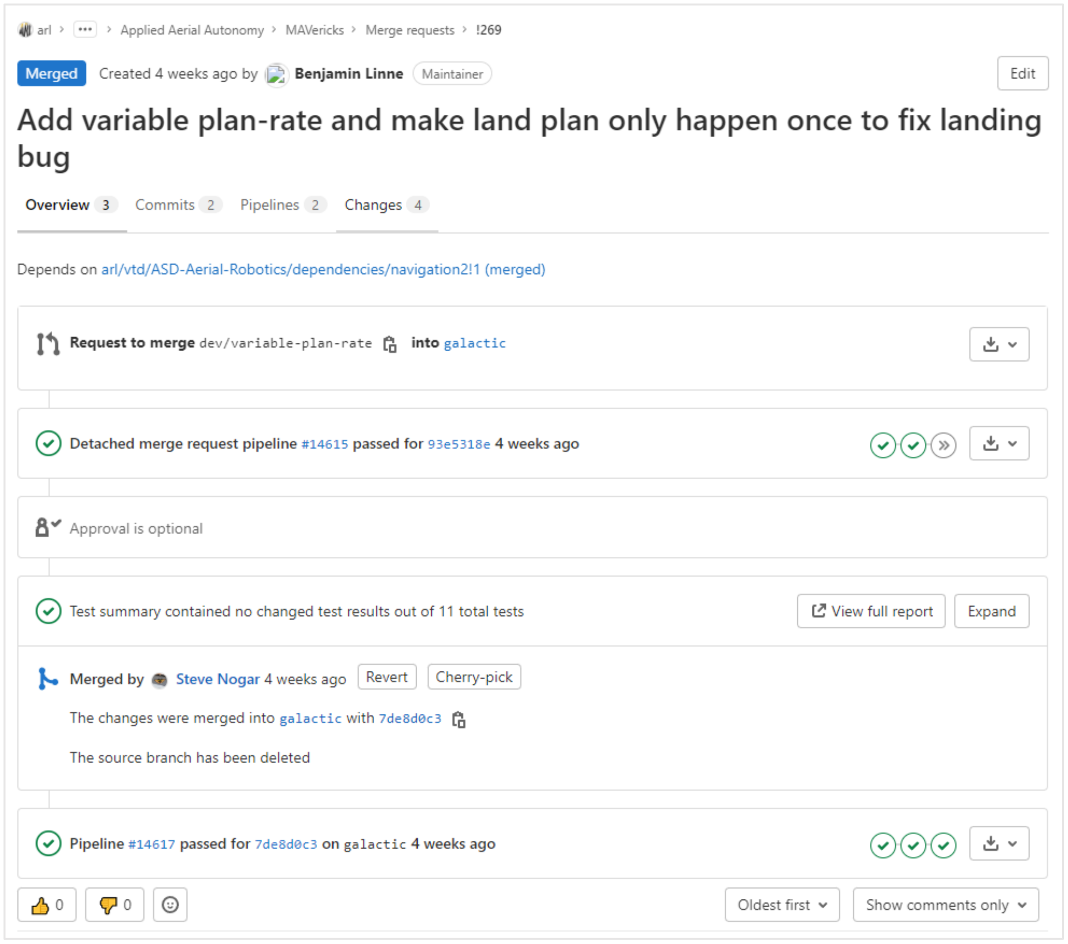
In this report, the foundation of the MAVericks CI/CD is described, then each stage in the pipeline is detailed along with several challenges that were overcome.

2. Pipeline

The MAVericks CI/CD is a pipeline architecture that is created utilizing the GitLab CI/CD framework. GitLab is a Git- or distributed-version control system hosting platform; it hosts the MAVericks code with a web user interface and interfaces with a CI/CD framework. This is made available to collaborators by the Sensor

Information Testbed Collaborative Research Environment (SITCORE), which is an Information Science Campaign initiative within ARL's Open Campus.*

To easily interact with CI/CD, the GitLab framework includes a user-interface that is customized to meet the needs of MAVericks. This allows users to easily view code and interact with CI/CD. One of the main features is the MR, which allows users to submit code changes, then a maintainer can view the specific changes, make comments, and accept or deny the changes. A sample MR is shown in Fig. 2, and the corresponding pipeline is shown in Fig. 3.



Created 4 weeks ago by Benjamin Linne Maintainer

Add variable plan-rate and make land plan only happen once to fix landing bug

Depends on arl/vtd/ASD-Aerial-Robotics/dependencies/navigation2!1 (merged)

Request to merge `dev/variable-plan-rate` into `galactic`

Detached merge request pipeline #14615 passed for `93e5318e` 4 weeks ago

Approval is optional

Test summary contained no changed test results out of 11 total tests

Merged by Steve Nogar 4 weeks ago

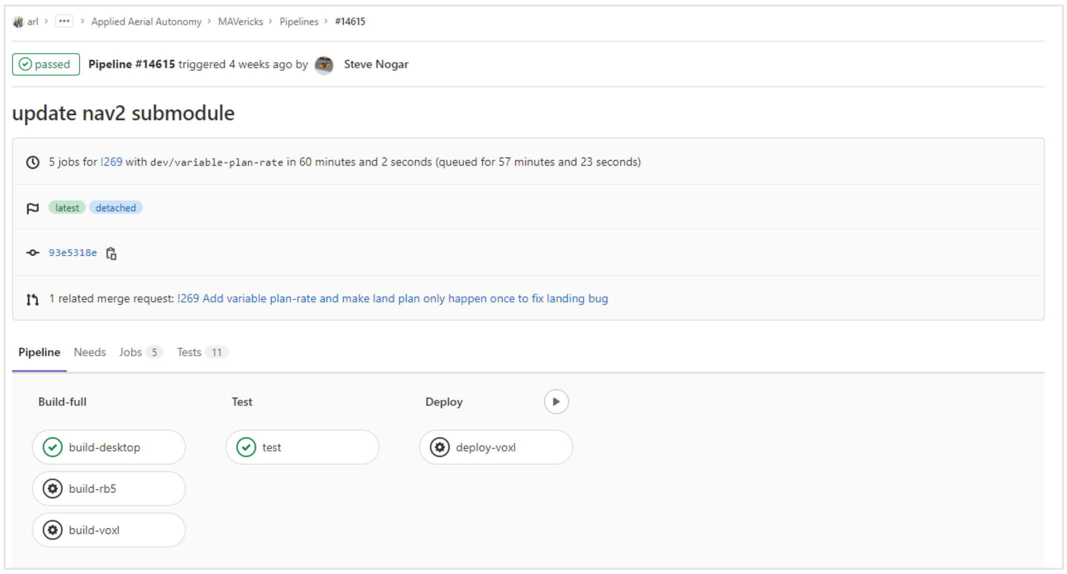
The changes were merged into `galactic` with `7de8d0c3`

The source branch has been deleted

Pipeline #14617 passed for `7de8d0c3` on `galactic` 4 weeks ago

Fig. 2 Sample MR

* The repository and pipeline are found at https://gitlab.sitcore.net/arl/vtd/ASD-Aerial-Robotics/mav_platform_r2.


arl › ··· › Applied Aerial Autonomy › MAVericks › Pipelines › #14615

passed **Pipeline #14615** triggered 4 weeks ago by Steve Nogar

update nav2 submodule

5 jobs for !269 with `dev/variable-plan-rate` in 60 minutes and 2 seconds (queued for 57 minutes and 23 seconds)

latest detached

`93e5318e`

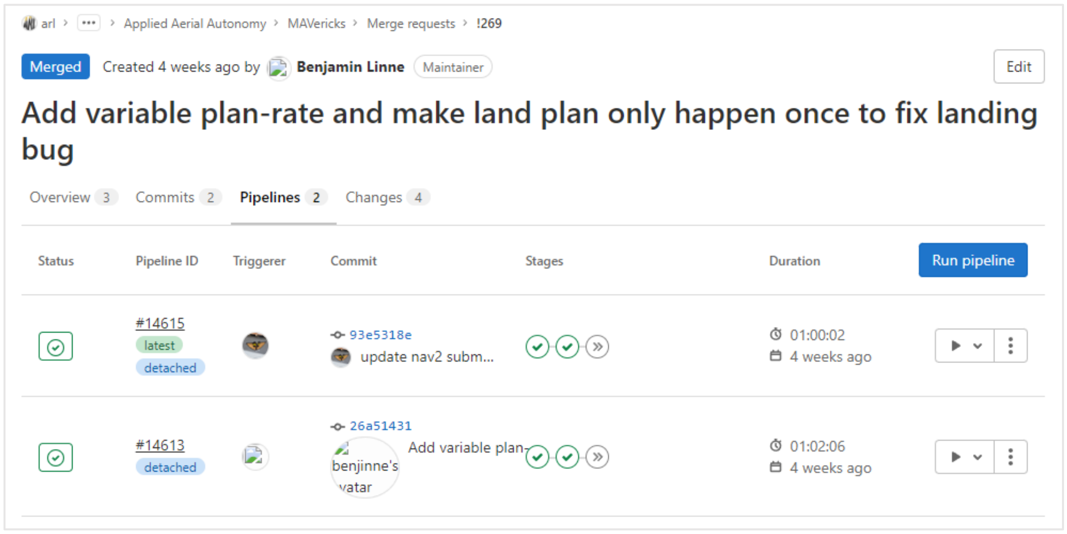
1 related merge request: !269 Add variable plan-rate and make land plan only happen once to fix landing bug

Pipeline Needs Jobs 5 Tests 11

Build-full	Test	Deploy
build-desktop	test	deploy-voxl
build-rb5		
build-voxl		

Fig. 3 Sample pipeline

After each change is made to the MR, a new pipeline is run, and gives the user another opportunity to fix any mistakes or improve the changes. An example list of pipelines for an MR is shown in Fig. 4.


arl › ··· › Applied Aerial Autonomy › MAVericks › Merge requests › !269

Merged Created 4 weeks ago by Benjamin Linne Maintainer Edit

Add variable plan-rate and make land plan only happen once to fix landing bug

Overview 3 Commits 2 Pipelines 2 Changes 4

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
✓	#14615 latest detached		93e5318e update nav2 subm...	✓ ✓ »	01:00:02 4 weeks ago
✓	#14613 detached		26a51431 Add variable plan benjinne's vatar	✓ ✓ »	01:02:06 4 weeks ago

Fig. 4 List of pipelines for an MR

3. Stages

Stages are run sequentially and only proceed after the stage succeeds. If any job in a stage fails, the pipeline fails, and the user is notified of what went wrong. The order and number of stages were created to minimize storage use and increase speed. The first stage is Build-Dependencies (i.e., build-deps), which is a conditional stage that only runs when needed to save time. It builds all of the dependencies that do not change frequently and is a single job that can be skipped if no modifications to the listed dependencies are made. Then, build-full runs three jobs in parallel. The first is for the desktop and every UAV type. Next, the test stage uses the desktop build to run the tests. Finally, the deploy stage is run and compresses each workspace into a tarball[*] that is ready to run a UAV.

4. Docker

To ensure that jobs are run in a clean environment, docker containers are used. Docker containers are virtual environments that are created for each job; they start from a clean install of the Ubuntu operating system. This guarantees that any new user who begins with a clean install will be able to run MAVericks and ensures that if a user makes changes that work on their machine, those changes will also work on any other machine. Docker also allows stages to divide work by creating images that are snapshots of a docker container, which can be passed from previous to subsequent stages.

5. Implementation Summary

The following sections cover the implementation details of the pipeline and the full MAVericks GitLab CI/CD configuration file.[†] For MAVericks specifics, a script for each job is in the "Docker/scripts" folder. This allows the ".gitlab-ci.yaml" file to focus on GitLab specifics such as setup and triggering jobs.

6. Build Dependencies

The build-deps stage consists of a single job that builds a list of large dependencies that are not frequently changed (i.e., only when modified).

In the ".gitlab-ci.yaml" file, the build-deps job is defined as follows:

```
build-desktop-dependencies:
  stage: build-deps
  rules:
    - changes:
    # Use file path for submodules
      - .gitlab-ci.yml
      - Docker/Dockerfile.desktop_deps.build
      - Docker/scripts/build_desktop_deps_docker.sh
```

[*] Tarball is a jargon term for a TAR archive—a group of files collected together as one.
[†] The ".gitlab-ci.yaml" file is available at: https://gitlab.sitcore.net/arl/vtd/ASD-Aerial-Robotics/mav_platform_r2/-/blob/galactic/.gitlab-ci.yml. Further GitLab specifics can be found on the GitLab documentation page: https://docs.gitlab.com/ee/ci/.

```
      - setup/**/*
      - src/communication/px4_msgs
      - src/communication/px4_ros_com
      - src/dependencies/**/*
      - src/perception/image_common/**/*
      - src/perception/image_pipeline/**/*
      - src/perception/sensor_msgs
      - src/localization/open_vins
  script:
    - ./Docker/scripts/build_desktop_deps_docker.sh
```

Under the rules section, this job is only run when changes to specific directories or files are changed. Then, if the job is run, the “build_desktop_deps_docker” script is executed as follows:

```

# Exits immediately if any line errors
set -e
# Reduces docker image siz
ln -sf .dockerignore.desktop .dockerignore
# Prints information relating gitlab runner storage to debug
build failure
# Due to storage constraints
./Docker/scripts/helper/debug_storage.sh
# Sets IMG variable depending on if job is run for a merge
request or after a
# Merge request into the default branch
if [ "$CI_COMMIT_REF_NAME" == "$CI_DEFAULT_BRANCH" ]; then
    IMG=${CI_REGISTRY_IMAGE}/${CI_DEFAULT_BRANCH}:desktop-deps
elif [ "$CI_PIPELINE_SOURCE" == "merge_request_event" ]; then

IMG=${CI_REGISTRY_IMAGE}/merge_requests/${CI_COMMIT_REF_SLUG}:des
ktop-deps
fi

source Docker/scripts/helper/volume_option.sh

# Running these steps outside of Dockerfile to avoid needing to
copy all files to desktop_deps image
# /mav_platform_r2 is used as the build directory for --symlink-
install to preserve paths
docker run ${VOLUME_OPTION} ros:galactic-ros-base \
/bin/bash -c 'export DEBIAN_FRONTEND=noninteractive \
&& export CI_DIR="/builds/arl/vtd/ASD-Aerial-
Robotics/mav_platform_r2/" \
&& apt-get update \
&& apt-get install -y apt-utils keyboard-configuration rsync \
&& rsync -a ${CI_DIR}/{.[!.],}* /mav_platform_r2 \
&& pushd /mav_platform_r2 \
&& cat setup/ros-key.txt | apt-key add - \
&& ./setup/install_desktop.sh \
&& colcon build --symlink-install --packages-up-to $(cat
setup/base_deps.txt) \
&& rsync -a /mav_platform_r2/{.[!.],}* ${CI_DIR} \
&& rm -rf /mav_platform_r2 \
&& echo $HOSTNAME > ${CI_DIR}/temp_deps_id.txt'
```

```
# Dependencies installed get committed, but data in volume or
CI_DIR does not
TEMP_DEPS=$(cat temp_deps_id.txt)
TEMP_IMAGE=$(docker commit ${TEMP_DEPS})
docker rm ${TEMP_DEPS}

# Build image with build and install
docker build --build-arg base_image=${TEMP_IMAGE} -t ${IMG} -f
./Docker/Dockerfiles/Dockerfile.desktop_deps.build .
docker push ${IMG}

./Docker/scripts/helper/debug_storage.sh
```

First, this script sets up the build by setting environmental variables and configuration files, then the build is run in a temporary docker image. In this image, the "install_desktop" script is run, which installs all necessary dependencies. Then, the build is initiated with "colcon". The "–packages-up-to" argument uses the "setup/base_deps.txt", which contains the list of dependency packages that are needed to build other packages—these typically take a long time to build and are not frequently updated. When this stage is skipped, it reduces the pipeline build time by over 45 min (how long the job takes to complete).

Finally, the docker build command executes the "Dockerfile.desktop_deps" docker script, which gets tagged with the image (IMG) variable and saved to the GitLab container registry to be used in the next stage. The docker script is shown as follows:

```
ARG base_image
FROM $base_image
WORKDIR /mav_platform_r2
COPY build build/
COPY install install/
ENTRYPOINT ["/ros_entrypoint.sh"]
CMD ["bash"]
```

This script copies the build and install directories into a new image that will get merged with the full build to create a fully built workspace.

7. Build-Full

The build-full stage contains the jobs to build desktop and UAV hardware, which includes the VOXL and RB5 hardware platforms. To support these platforms, three jobs are run in parallel: build-desktop, build-rb5, and build-voxl. First, each job builds all packages and fails if any package has an error building, then the user is notified which package failed before proceeding. After the build is successful, the build-desktop job creates an image used for the test stage. It can also be used for further experimenting. The build-rb5 and build-voxl jobs then build compressed workspaces in the form of a tarball. This process dramatically reduces the time needed to deploy MAVericks to less than half an hour on these hardware targets since building the complete workspace on a VOXL or on a developer machine can take several hours. This allows developers to continue preparing for flight tests while their workspace is being built.

In the ".gitlab-ci.yaml" file, the build-desktop job is defined as follows:

```
build-desktop:
  stage: build-full
  script:
    - ./Docker/scripts/build_desktop_docker.sh
```

This job runs on every pipeline and calls the "build_desktop_docker" script, which contains the following:

```
set -e
ln -sf .dockerignore.desktop .dockerignore

./Docker/scripts/helper/debug_storage.sh

# Sets BASE_IMG and FULL_IMG
source Docker/scripts/helper/tag_helper.sh "desktop-full"
# Sets VOLUME_OPTION
source Docker/scripts/helper/volume_option.sh

# This build could fail during a MR pipeline when a different MR
merges with DEFUALT_BRANCH and updates desktop-deps
# If this begins to cause problems, we should consider combining
the build_deps and build_full stages
# Another fix is to merge with master to fix any conflicts
docker run --rm --pull always ${VOLUME_OPTION} ${BASE_IMG} \
/bin/bash    -c    'export    CI_DIR="/builds/arl/vtd/ASD-Aerial-
Robotics/mav_platform_r2/" \
&& rsync -a ${CI_DIR}/{.[!.],}* /mav_platform_r2 \
&& pushd /mav_platform_r2 \
&&   colcon   build   --symlink-install   --packages-skip   $(cat
setup/base_deps.txt) \
&& rsync -a /mav_platform_r2/{.[!.],}* ${CI_DIR} \
&& rm -rf /mav_platform_r2'

# Build final image with needed files for final image
docker build --build-arg base_image=${BASE_IMG} -t ${FULL_IMG} -f
./Docker/Dockerfiles/Dockerfile.desktop.build .
docker push ${FULL_IMG}

./Docker/scripts/helper/debug_storage.sh
```

Similar to build-deps, this script first sets up the build by setting environment variables and configuration files, then the build starts with the build-deps image, which comes from the optional stage or from the default branch if it was not run. Then the "colcon" build begins with a partially built workspace and skips all of the prebuilt packages with the “–packages-skip” argument. Finally, the docker image is built using “Dockerfile.desktop.build” as shown:

```
ARG base_image
FROM $base_image
WORKDIR /mav_platform_r2
COPY env env/
COPY setup setup/
COPY src src/
COPY build build/
COPY install install/
CMD ["sh -c 'strip --remove-selection=.note.ABI-tag /lib/x86_64-
linux-gnu/libQtCore.so.5'"]
ENTRYPOINT ["/ros_entrypoint.sh"]
CMD ["bash"]
```

This script is very similar to the build-deps docker file. However, this one copies the source code along with the build and install so that this image contains everything needed to run MAVericks.

In the “.gitlab-ci.yaml” file, the build-voxl is defined as follows:

```
build-voxl:
  stage: build-full
  tags:
    - arm
  rules:
  - if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'
  - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    when: manual
    allow_failure: true # To allow skip
  artifacts:
    reports:
      dotenv: deploy.env
    expire_in: 4 week
    paths:
      - mavericks_voxl.tar.gz
  script:
    - ./Docker/scripts/build_voxl_docker.sh
```

The build-voxl job is unique because it uses the arm tag, which is used because it will pick an arm64 builder that allows the build to speed up so that it goes from taking several hours to just 30 min. When a regular amd64 runner is running an arm64 image, there is a lot of overhead translating between amd64 and arm64.

The rules tag specifies that this job is always run on the default branch; however, it is an optional job for MRs to reduce the number of unnecessary builds being run since there are only two GitLab runners available. The number of tarballs created is also reduced, which saves space on the GitLab server.

Next, the output is an artifact called “mavericks_voxl.tar.gz” that is set to expire in 4 weeks (i.e., it will be deleted). In comparison to the build-rb5 job, it is almost identical except that build-rb5 has a different artifact name and calls a different script. Finally, the “build_voxl_docker” script is run; it contains the following:

```
set -e
ln -sf .dockerignore.voxl .dockerignore

VOXL_DEPLOY=registry.gitlab.sitcore.net:443/arl/vtd/asd-aerial-
robotics/dependencies/voxl-deploy:galactic-3-2-image

# Sets BASE_IMG and FULL_IMG
source Docker/scripts/helper/tag_helper.sh "voxl-full"
# Sets VOLUME_OPTION
source Docker/scripts/helper/volume_option.sh

# Get px4_ros_com generated files since voxl image can't generate
TEMP_BASE=$(docker run -d --pull always ${BASE_IMG})
mkdir -p build/px4_ros_com/
docker    cp    $TEMP_BASE:/mav_platform_r2/build/px4_ros_com/src
build/px4_ros_ com/
docker rm $TEMP_BASE

./Docker/scripts/helper/debug_storage.sh

docker run --rm --pull always ${VOLUME_OPTION} ${VOXL_DEPLOY} \
/bin/bash    -c    'export    CI_DIR="/builds/arl/vtd/ASD-Aerial-
Robotics/mav_platform_r2/" \
&& rsync -a ${CI_DIR}/{.[!.],}* /data/mav_ws \
&& pushd /data/mav_ws \
&& export LD_LIBRARY_PATH=/usr/local/lib \
&& source /opt/ros/galactic/setup.bash \
&& export MAVERICKS_PLATFORM=VOXL \
&& colcon build --symlink-install \
--cmake-args  -DBUILD_TESTING=OFF  -DCMAKE_BUILD_TYPE=Release  -
DCMAKE_SHARED_LINKER_FLAGS='-latomic'  -DCMAKE_EXE_LINKER_FLAGS='-
latomic' \
--packages-skip $(cat setup/voxl_skip_deps.txt) \
&& rsync -a /data/mav_ws/{.[!.],}* ${CI_DIR} \
&& rm -rf /data/mav_ws'

tar -czf mavericks_voxl.tar.gz install/ build/

./Docker/scripts/helper/debug_storage.sh

echo "FULL_VOXL_JOB_ID=${CI_JOB_ID}" >> deploy.env
```

This job is like the build-desktop job, except that it uses the “arm64 VOXL_DEPLOY” IMG to build. Then, since the VOXL_DEPLOY IMG is unable to build “px4_ros_com” correctly, the “px4_ros_com/src” files are copied from the build-deps. These are files generated in the “src directory” as part of the build process, and without them, building the “px4_ros_com” package will fail using the VOXL_DEPLOY IMG.

Before the build begins, a temporary container is run from the VOXL_DEPLOY IMG. In this image, the source is copied using "rsync", into the "/data/mav_ws" directory since that is where it will live on the VOXL. The build needs to run in this directory because the build is run with the "–symlink-install." If MAVericks is run in another directory, the symlinks will not point to the correct directories.

Then, the build is started and the "voxl_skip_deps" are provided to the "–packages-skip" argument to prevent building unnecessary packages such as simulation or visualization packages.

Finally, the tarball is created by compressing the build/ and install/ directories, and the "deploy.env" artifact is used as an identifier for the deploy stage.

8. Test

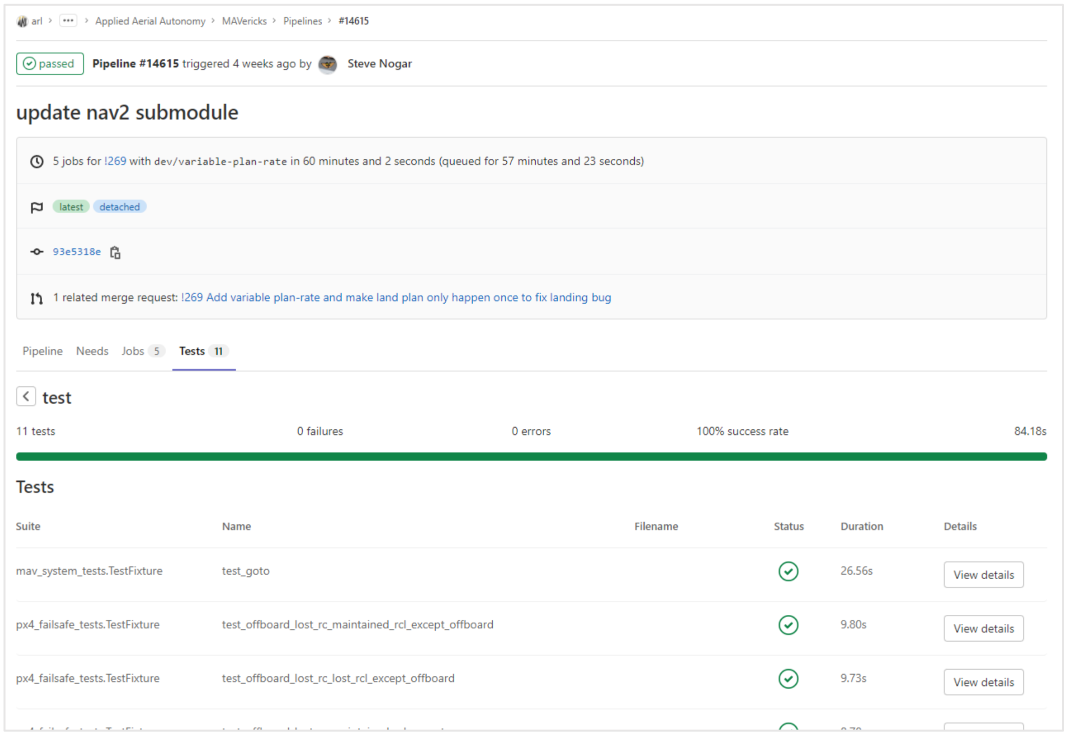
Before the deploy stage is run, a series of tests are run before merging any changes and releasing a build. These tests include testing behaviors and fail-safes. The behavior tests currently only have a go-to test, and the fail-safe tests include combinations of off-board lost and Remote Control (RC) lost to ensure the correct response is executed. This job is defined as follows in the ".gitlab-ci.yaml" file:

```
test:
  stage: test
  variables:
    GIT_STRATEGY: fetch
    GIT_SUBMODULE_STRATEGY: none
  # Ensure test jobs are run sequentially to prevent race condition
  resource_group: test
  script:
    - ./Docker/scripts/tests.sh
  artifacts:
    when: always
    reports:
      junit:
        - mav_report.xml
        - px4_report.xml
```

To save the test job about 10 min, a full clone of the workspace is not needed; therefore, the Git variables are set to just perform a Git fetch. This way at least the docker scripts are available to run the build-desktop image and the tests.

The resource group ensures that tests from other pipelines are not run at the same time to avoid conflicts between docker containers. This may not be needed, but it attempts to fix an issue with tests failing.

Finally, after the test script is run, the test results are saved and exported as JUnit files to better represent them in the GitLab web interface. Thus, if the job fails users can easily identify the tests that need work. An example is shown in Fig. 5.


arl › ••• › Applied Aerial Autonomy › MAVericks › Pipelines › #14615

passed **Pipeline #14615** triggered 4 weeks ago by Steve Nogar

update nav2 submodule

5 jobs for !269 with `dev/variable-plan-rate` in 60 minutes and 2 seconds (queued for 57 minutes and 23 seconds)

latest detached

93e5318e

1 related merge request: !269 Add variable plan-rate and make land plan only happen once to fix landing bug

Pipeline Needs Jobs 5 **Tests 11**

test

11 tests 0 failures 0 errors 100% success rate 84.18s

Tests

Suite	Name	Filename	Status	Duration	Details
mav_system_tests.TestFixture	test_goto		✓	26.56s	View details
px4_failsafe_tests.TestFixture	test_offboard_lost_rc_maintained_rcl_except_offboard		✓	9.80s	View details
px4_failsafe_tests.TestFixture	test_offboard_lost_rc_lost_rcl_except_offboard		✓	9.73s	View details

Fig. 5 Example list of successful tests

The test script contains the following:

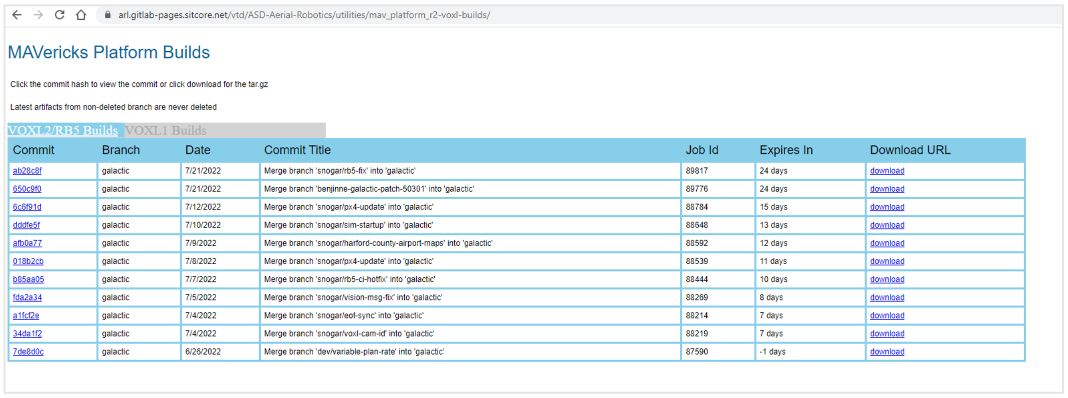
```
set -e

if [ "$CI_COMMIT_REF_NAME" == "$CI_DEFAULT_BRANCH" ]; then
    IMG=${CI_REGISTRY_IMAGE}/${CI_DEFAULT_BRANCH}:desktop-full
elif [ "$CI_PIPELINE_SOURCE" == "merge_request_event" ]; then

IMG=${CI_REGISTRY_IMAGE}/merge_requests/${CI_COMMIT_REF_SLUG}:des
ktop-full
fi

# Sets VOLUME_OPTION
source Docker/scripts/helper/volume_option.sh

# Fail script if any test fails, but run all tests. See
https://unix.stackexchange.com/a/596968
docker run --pull always ${VOLUME_OPTION} --rm ${IMG} /bin/bash -c
\
'pushd /mav_platform_r2; \
source env/sim.sh; \
test_dir='src/mav_system_tests/mav_system_tests'; \
report_dir='/builds/arl/vtd/ASD-Aerial-Robotics/mav_platform_r2';
\
ss=0; \
launch_test          --junit-xml          $report_dir/mav_report.xml
$test_dir/mav_system_tests.py || ((ss++)); \
launch_test          --junit-xml          $report_dir/px4_report.xml
$test_dir/px4_failsafe_tests.py || ((ss++)); \
exit $ss'
```

First, the correct desktop-full image is selected depending on whether the job is running on the main branch or an MR. Then, the tests are run using the "ros2 launch_test" command. Currently, only two integration tests are run. These include the "mav_system_tests" and "px4_failsafe_tests". The "mav_system_tests" launch MAVericks and send a go-to mission, which succeeds if the way point is achieved. The "px4_failsafe_tests" simulate a series of fail-safes such as RC lost or off-board lost, then ensures the correct response is executed. These two integration tests ensure a minimal functionality is achievable; however, more tests are needed to ensure every future change maintains a high level of functionality.

If both tests are successful, the job succeeds. If the first test fails, the second one is still run since the commands are executed together using a semicolon instead of an ampersand. If the "$ss" variable increments are above zero the job fails; however, all reports are saved and viewable.

9. Deploy

A complete build of MAVericks takes several hours for the UAV hardware; therefore, it is extremely useful to have prebuilt versions available to download at any time. The result of build-full creates tarballs that can easily be extracted on UAVs and are immediately ready to run the code. In the deploy stage, the deploy-voxl job exposes these tarballs to an easy-to-view web page for users to access and download. This web page contains the commit, branch, date, commit title, job ID, expiration time, and download URL for each available tarball (Fig. 6).


arl.gitlab-pages.sitcore.net/vtd/ASD-Aerial-Robotics/utilities/mav_platform_r2-voxl-builds/

MAVericks Platform Builds

Click the commit hash to view the commit or click download for the tar.gz

Latest artifacts from non-deleted branch are never deleted

VOXL2/RB5 Builds VOXL1 Builds

Commit	Branch	Date	Commit Title	Job Id	Expires In	Download URL
ab28c8f	galactic	7/21/2022	Merge branch 'snogar/rb5-fix' into 'galactic'	89817	24 days	download
650c9f0	galactic	7/21/2022	Merge branch 'benjinne-galactic-patch-50301' into 'galactic'	89776	24 days	download
6c6f91d	galactic	7/12/2022	Merge branch 'snogar/px4-update' into 'galactic'	88784	15 days	download
dddfe5f	galactic	7/10/2022	Merge branch 'snogar/sim-startup' into 'galactic'	88648	13 days	download
afb0a77	galactic	7/9/2022	Merge branch 'snogar/harford-county-airport-maps' into 'galactic'	88592	12 days	download
018b2cb	galactic	7/8/2022	Merge branch 'snogar/px4-update' into 'galactic'	88539	11 days	download
b85aa05	galactic	7/7/2022	Merge branch 'snogar/rb5-ci-hotfix' into 'galactic'	88444	10 days	download
fda2a34	galactic	7/5/2022	Merge branch 'snogar/vision-msg-fix' into 'galactic'	88269	8 days	download
a1fcf2e	galactic	7/4/2022	Merge branch 'snogar/eot-sync' into 'galactic'	88214	7 days	download
34da1f2	galactic	7/4/2022	Merge branch 'snogar/voxl-cam-id' into 'galactic'	88219	7 days	download
7de8d0c	galactic	6/26/2022	Merge branch 'dev/variable-plan-rate' into 'galactic'	87590	-1 days	download

Fig. 6 VOXL and RB5 tarball download page

After all tests succeed, the job runs in the Python 3.7.0 IMG since only Python is needed. The "before_script" section is skipped because docker is not needed and fails the job since it is not included in the Python 3.7.0 IMG. Then, like the build-voxl job, the rules are set to optionally run for MRs but are always run for the default branch. Also, the Git variables and "resource_group" are set like the test job for speed and to prevent conflicts between pipelines, respectively.

```
deploy-voxl:
  stage: deploy
  image: "python:3.7"
  # Remove before_script since docker command will fail in python
image
  before_script: []
  rules:
    - if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
      when: manual
      allow_failure: true # To allow skip
  variables:
    GIT_STRATEGY: fetch
    GIT_SUBMODULE_STRATEGY: none
  # Ensure deploy-voxl jobs are run sequentially to prevent race
condition
  resource_group: deploy
  script:
    - ./Docker/scripts/deploy_platforms.sh
```

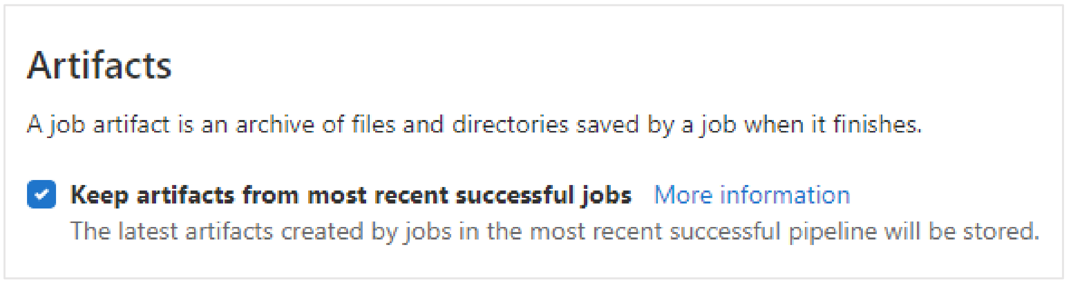
The last part is the script that runs the "deploy_platforms" scripts, which displays the build artifacts on a web page. This script contains the following:

```
#! /bin/bash
set -e

pip install python-csv python-dateutil

VOXL_BUILDS_REPO="https://root:${BOT_ACCESS_TOKEN}@${CI_SERVER_HO
ST}/arl/vtd/ASD-Aerial-Robotics/utilities/mav_platform_r2-voxl-
builds.git"
VOXL_BUILDS_DIR=artifact/mav_platform_r2-voxl-builds/
git clone "$VOXL_BUILDS_REPO" "$VOXL_BUILDS_DIR"

RB5_LOG_FILE=${VOXL_BUILDS_DIR}public/rb5_artifact_log.txt
VOXL_LOG_FILE=${VOXL_BUILDS_DIR}public/voxl_artifact_log.txt


if [[ -n "$FULL_RB5_JOB_ID" ]]; then
    ./Docker/scripts/helper/update_artifact_log.py
"${CI_COMMIT_SHA}"  "${CI_COMMIT_REF_NAME}"  "${CI_COMMIT_TITLE}"
"${FULL_RB5_JOB_ID}" "${RB5_LOG_FILE}"
    cp ${RB5_LOG_FILE}.tmp ${RB5_LOG_FILE}
    rm ${RB5_LOG_FILE}.tmp
    COMMIT_RB5=1
fi

if  [[ -n "$FULL_VOXL_JOB_ID" ]]; then
    ./Docker/scripts/helper/update_artifact_log.py
"${CI_COMMIT_SHA}"  "${CI_COMMIT_REF_NAME}"  "${CI_COMMIT_TITLE}"
"${FULL_VOXL_JOB_ID}" "${VOXL_LOG_FILE}"

    cp ${VOXL_LOG_FILE}.tmp ${VOXL_LOG_FILE}
    rm ${VOXL_LOG_FILE}.tmp
    COMMIT_VOXL=1
fi

cd ${VOXL_BUILDS_DIR}
git config --global user.name "${GITLAB_USER_NAME}"
git config --global user.email "${GITLAB_USER_EMAIL}"

if [[ "${COMMIT_RB5}" -eq "1" ]];then
    git add public/rb5_artifact_log.txt
fi
if [[ "${COMMIT_VOXL}" -eq "1" ]];then
    git add public/voxl_artifact_log.txt
fi
git commit -m "update log for job# ${CI_JOB_ID}"
git                                                          push
https://root:$BOT_ACCESS_TOKEN@$CI_SERVER_HOST/arl/vtd/ASD-
Aerial-Robotics/utilities/mav_platform_r2-voxl-builds.git
```

This script works by publishing the download URLs for the tarballs and descriptions to the GitLab web page repository.[*] After 28 days (or when a branch is deleted), the respective tarballs are deleted. An exception is made if it is the most

[*] Accessible at https://arl.gitlab-pages.sitcore.net/vtd/ASD-Aerial-Robotics/utilities/mav_platform_r2-voxl-builds/.

recent successful build for a branch. This option in the GitLab CI/CD settings is shown in Fig. 7.


Artifacts

A job artifact is an archive of files and directories saved by a job when it finishes.

✓ **Keep artifacts from most recent successful jobs** More information

The latest artifacts created by jobs in the most recent successful pipeline will be stored.

Fig. 7 GitLab artifact configuration

If any tarballs have expired, they are removed from the web page on the next update. If no new updates have been made after a tarball has expired, it will show up as expired since the "expires_in" column is calculated from the viewers browser in JavaScript.

10. Future Improvements

The current size of the build-deps and build-desktop images are 4 and 11 GB compressed, respectively. As they get larger CI/CD runs more slowly (since loading and saving images currently takes several minutes) and more potential exists for GitLab storage constraints to be exceeded. It would be wise to attempt to reduce these sizes by either eliminating unnecessary packages in MAVericks or by optimizing the docker scripts.

In the testing stage, there are currently no unit tests or feature testing. By adding more tests and increasing the code coverage, developers will save time and effort when developing new features. This is done by preventing new bugs from being introduced when new features are being developed. Also, it is not currently possible to run graphical simulations for more extensive integration testing. It should be possible to run them without a monitor attached (i.e., headless mode); however, this still needs to be explored. Unit tests for behavior trees are possible from the open-source examples; however, none have been created for the MAVericks behaviors. Additional tests are recommended; however, their duration must remain reasonable.

Another way to increase pipeline speed is by only running build-deps on "commits" (i.e., how Git refers to changes) instead of for the entire MR. This would be useful since MRs that are open for a long time may only have one small dependency change, but many unrelated changes. Unfortunately, after an MR has modified a dependency, every subsequent commit must run build-deps, which adds unnecessary delay. This may not be possible depending on the GitLab configuration but could be a feature request for GitLab to implement.

11. Conclusion

CI/CD is a powerful tool that allows developers to effectively collaborate by way of a pipeline, which is run after each modification to MAVericks. It dramatically increases the reliability of the software, while reducing the amount of manual testing needed by software engineers to verify new changes. These tools allow for distributed development between ARL and other DEVCOM centers, industry, and academia.

CI/CD may not always reduce build times, but it allows developers to free up their machine for developmental work since building is usually a long process that consumes all available processing power. It also enables a developer to build multiple experimental builds without needing to maintain built workspaces on their machine.

List of Symbols, Abbreviations, and Acronyms

ARL	Army Research Laboratory
build-deps	Build-Dependencies
CI/CD	continuous integration/continuous delivery
DEVCOM	US Army Combat Capabilities Development Command
DOD	Department of Defense
IMG	image
MR	Merge Request
RC	Remote Control
SITCORE	Sensor Information Testbed Collaborative Research Environment
UAV	unmanned autonomous vehicle
URL	uniform resource locator

1	DEFENSE TECHNICAL
(PDF)	INFORMATION CTR
	DTIC OCA
1	DEVCOM ARL
(PDF)	FCDD RLD DCI
	TECH LIB
1	DEVCOM ARL
(PDF)	FCDD RLW TD
	B LINNE