NDSU NORTH DAKOTA STATE UNIVERSITY

Spaceport Command and Control System User Interface Testing

Center:	Kennedy Space Center
Program:	NE-ES Spring Session
Date:	4/22/2016
Student Name:	Jacob Huesman
Academic Level:	Sophomore
Academic Major:	Electrical Engineering
Academic Institution:	North Dakota State University
Mentor Name:	Jamie Szafran
Mentor Job Title:	Computer Scientist/Engineer
Org Code/Branch:	NE-ES
Division:	Electrical Division
Directorate:	Software

The Spaceport Command and Control System will be the National Aeronautics and Space Administration's newest system for launching commercial and government owned spacecraft. It's a large system with many parts all in need of testing. To improve upon testing already done by NASA engineers, the Engineering Directorate, Electrical Division (NE-E) of Kennedy Space Center has hired a group of interns each of the last few semesters to develop novel ways of improving the testing process.

Nomenclature

IDE	=	Integrated Development Environment is a program that enhances the programming process
Java	=	General purpose programming language designed to have as few implementation dependencies as possible
Jenkins[1]	=	Automation server used for building and testing software projects
Jython	=	Java implementation of Python
KSC	=	Kennedy Space Center is the NASA center tasked with the assembly and launch of rockets, in addition to numerous research activities
LCC	=	Launch Control Center is the building in KSC utilized for controlling launches
LCS	=	Launch Control System is the KSC software system used for remotely controlling launches
NASA	=	National Aeronautics and Space Administration
OpenCV[2]	=	Open source image recognition library
Portal Workstation	=	Computer located in firing room used for accessing SCCS
Python	=	High level, interpreted programming language
Robot Framework[3]	=	Generic test automation framework used for running tests and generating test reports
SikuliX[4]	=	Software package used for automating keyboard strokes and mouse clicks
SCCS	=	Spaceport Command and Control System consists of the Launch Control System and the Kennedy Ground Control System, which together are the launch system for the SLS
SLS	=	Space Launch System is NASA's newest launch vehicle
Tesseract[5]	=	Open source optical character recognition tool
Workstation	=	Computer used for developing software
User Interface	=	The portion of a computer application that interacts with the user

I. Introduction

This internship focused on the development of automated user interface testing tools in collaboration with other interns and full time engineers. Normally, in order to test a user interface, an engineer would sit at a computer and click through or type in very specific test steps. The engineer would then verify that the software under test behaved as the test steps indicate it should. Automating the test steps has the potential to free up time on the part of the engineer and help eliminate human error by ensuring the test steps are performed consistently.

II. Objectives

The primary goal of the internship was to figure out how to integrate SikuliX and Robot Framework into a standalone package that could be placed on any Portal Workstation in the Launch Control Center Firing Room to test the Launch Control System User Interface. There were also several secondary goals that had the potential to improve the functionality of the SikuliX Robot Framework package. SikuliX has an experimental feature that allows for the use of text recognition in place of the default recognition process, which is image recognition[4]. Enabling and configuring that feature, which became a secondary goal, had the potential to drastically reduce the size of the SikuliX Robot Framework test packages by reducing the number of images that had to be included in the package. The text recognition feature also had the potential of improving the reliability of the package by reducing the error introduced by manually taking a snapshot of a user's display and cropping out the portion to be clicked on. An additional goal was to make the package deployable through the Jenkins build and deployment server. This would allow for hands free testing of the user interfaces.

III. Approach

To begin, we were given the work of the interns from last semester. They had primarily worked on researching existing test frameworks that could be used to satisfy the project objectives. They had come to the conclusion that Robot Framework with a SikuliX library would be the best fit. Robot Framework has a unique keyword based syntax that allows for near natural language programming. In addition it generates reports that are easy to read and scalable for large test sets. SikuliX allows for the automation of mouse clicks and keyboard strokes through the use of OpenCV image recognition[4]. Combining the two into a standalone package would be ideal for the automation of user interface tests.

We had several requirements that needed to be satisfied for the standalone package. The package itself and its dependencies can be visualized using Figure I, located in the Appendix. First, it had to be independent of the workstation. The process for getting software installed on Portal Workstations is quite extensive. Our mentors wanted to avoid the extra work if at all possible. This meant that we could not do the default installs of Robot Framework and SikuliX. The software had to be installed in a separate folder, or package, with no dependencies on the local machine that couldn't be expected to be satisfied by any Portal Workstation or SCCS Red Hat Workstation. Secondly, the packages had to work on the Red Hat Enterprise Linux operating system installed on all workstations. Robot Framework and SikuliX, being open source, are normally developed and tested on the most recent version of the Ubuntu Linux operating system; therefore the software wasn't tested on operating systems that place an emphasis on package stability, like Red Hat. This slightly complicated the installation as additional dependencies needed to be compiled and installed before Robot Framework and SikuliX could be installed. Third the package needed to include the SikuliX and Robot Framework IDEs. The Robot Framework IDE unfortunately was dependent upon a newer version of Python than what was installed by default on the workstations. Unlike the other dependencies, an updated version of Python could not be installed in a machine independent way. The solution we came up with was to create two separate packages. One would be machine independent, containing SikuliX and Robot Framework, which would be able to run the user interface tests. The other would contain the IDEs for developing the tests, but would have a dependency that required the local workstation to have an updated version of Python installed. Lastly, the entire process had to be documented so as to allow for future developers to be able to easily reproduce and update the package. This was implemented by placing a README file in the base directory of each package.

During the installation process, problems/bugs were discovered and fixed. Soon after installation we found that the Robot Framework IDE could not start up when running SikuliX based tests. The Robot Framework IDE, written in Python, was calling a method, os.getpid(). This method worked fine for Python based tests, however our tests, due to SikuliX requirements, were written in Jython. Unfortunately, os.getpid() currently has several bugs in its Jython implementation. Another intern, Nicole Maguire, wrote a patch in Java that could be used as an alternative to the method call causing trouble. The Robot Framework library that was to be used for integrating SikuliX and Robot Framework contained some code duplication that caused the library to fail to run. We fixed this by removing the duplicate code and then suggesting the fix to the library author through GitHub[6]. The library additionally didn't have a license readily available. This created an implicit default copyright on the code, meaning that the author retained all rights to the source code and that nobody else could reproduce, distribute, or create a derivative work of the library[7]. Without a more open license, we would have needed to refrain from using the library and develop our own. Developing our own library would have been by no means impossible, but it would also not have been ideal knowing that there was one already in existence. The author of the library was contacted and agreed to open up the software under the MIT License. Once the license was opened up, the library was expanded to include some previously missing SikuliX functionality.

After the two SikuliX/Robot Framework packages were set up and their functionality verified, we moved on to creating working examples of package functionality. This was done by writing a series of Robot Framework test cases following a real set of test instructions used to test the Launch Control System User Interface. Margaret Dube and I introduced two competing approaches in the interest of figuring out the best way of writing test cases. Margaret approached writing the test cases using syntax consisting mostly of default Robot Framework keywords. By doing so she maintained a certain level of simplicity, easily readable by those familiar with the Robot Framework syntax. Trying to take things a step further, I attempted to write generic keywords that matched the syntax currently contained in the test cases used by NASA engineers. My approach ended up generating a lot of underlying complexity in the generic keywords, so we decided to go with Margaret's approach for the remainder of the test cases. Several generic keywords were useful enough to be kept for future test cases.

The next step was to get text recognition to work. SikuliX uses a program called Tesseract for image recognition[4]. On our first attempt to utilize the feature, the results were very poor. Tesseract was recognizing the character G as C, L as I, and was also inserting random spaces. There was a process available to "train" Tesseract, so we attempted that as well. The training improved the accuracy of the text recognition, but it still wasn't recognizing the text accurately enough to be considered as an alternative to the image recognition. This was mostly due to Tesseract having been designed for high resolution images (the developers suggest around 300 DPI[5]); our screenshots were far lower resolution. Jason Kapusta, one of our mentors, suggested that since we know the font, size, and color of the background being used by the dashboard, we might be able to generate an image that could be used by SikuliX for interacting with the User Interface. This technique would allow us to save space in the test folders by not having to store a large number of screenshots and it would allow us to sidestep the Tesseract text recognition process entirely. While Margaret and I were working on Tesseract, Nicole developed a Java program to generate the image. After addressing some font issues, it worked perfectly. The solution Nicole developed worked significantly better than the screenshot based approach and the Tesseract based approach. We therefore accepted it as the solution of choice, with image recognition being used as a backup.

IV. Results

Our team successfully managed to get a user interface testing framework up and running. The package containing the framework can be run on any SCCS Red Hat workstation. Looking to improve the functionality, our team developed custom keywords and applications that sped up the process of developing automated User Interface tests. We created extensive documentation and examples that can be used by future interns and engineers for both improving the package and developing tests. At the time of this writing, there is one remaining task, which is getting the test packages to deploy automatically using the Jenkins deployment server. With five weeks left in the internship, I expect we'll get it up and running.

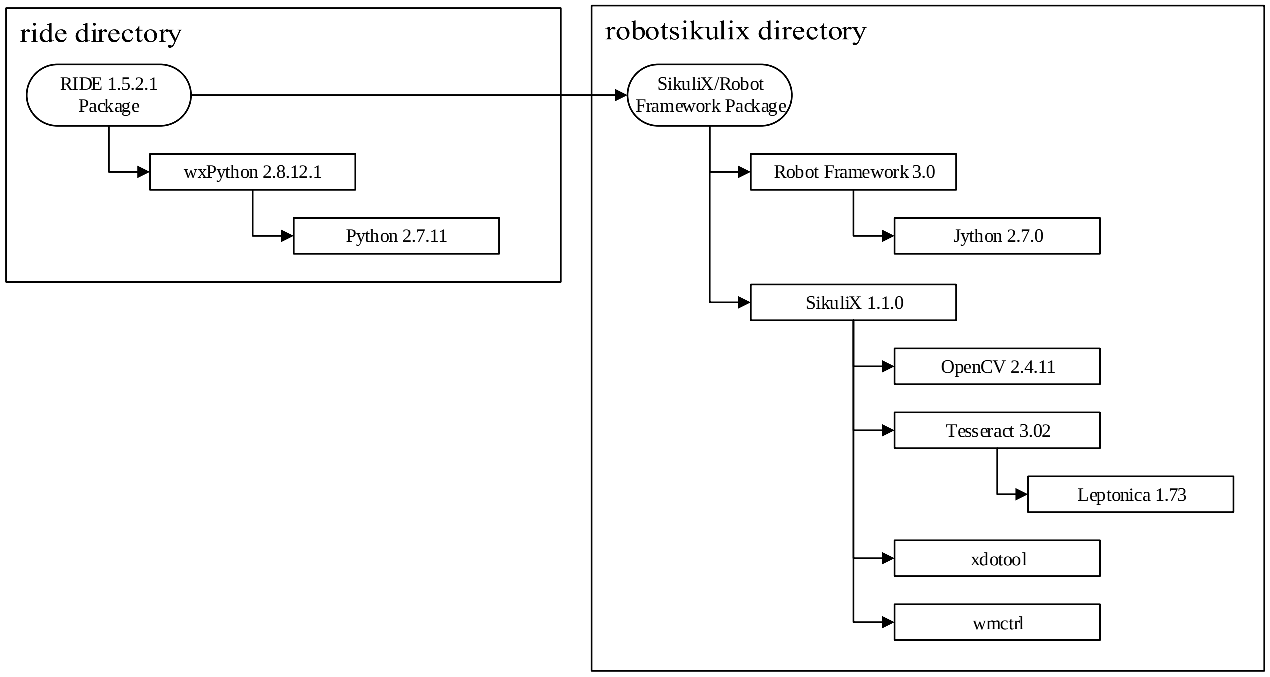
Acknowledgements

I'd like to thank Kyle Besser for sitting for hours in front of a computer screen with me, trying to figure out how to compile Tesseract and numerous other software packages. Thanks to Margaret Dube for knocking out the Level 5 example tests, while I was working on the generic keywords. Joshua Connolly deserves praise for spearheading the SWiT tests as the rest of us were trying to figure out Tesseract. When she wasn't busy with Display Services and Framework tests, Nicole Maguire had a tendency of stepping in and quickly finding novel solutions to serious headaches. I'm not sure where we would be without her. They were an awesome team to work with this semester.

The internship wouldn't have been possible without the full time employees working hard to keep us productive. The education office put up a ton of work getting us out here, making sure we were paid, scheduling social events, tours and training. Thank you Rose and Grace. Our technical points of contact, Jason and Paul, were a great source of direction and help throughout the entire internship. Our mentors, Jamie and Caylyne put in a ton of legwork to make sure that we were able to immediately get started upon arrival. I think I remember hearing that it used to take a month for an intern to become fully set up and done with mandatory training, it took less than a week with them. Will Denis was particularly tolerant of my numerous intrusions into his work schedule. Oscar kept us out of trouble (impressive considering the group of interns under his supervision) and pushed things through that otherwise would have been impossible. There were numerous other full time employees that helped us as well; thank you all.

Appendix

Figure 1.



References

Software

[1] Jenkins Community, "Jenkins", *jenkins.io,* May 31, 2015
[2] Itseez, "OpenCV", *opencv.org,* Dec. 21, 2015
[3] Robot Framework Community, "Robot Framework", *robotframework.org, github.com/robotframework/robotframework,* Dec. 31, 2015
[4] RaiMan, "SikuliX", *sikulix.com,* Oct. 7, 2015
[5] Smith, R., "Tesseract", *github.com/tesseract-ocr/tesseract,* Feb. 16, 2016
[6] jaredfin, "SikuliXRobotLibrary", *github.com/jaredfin/SikuliXRobotLibrary,* Feb. 20, 2016

Licensing

[7] GitHub, "No License", *choosealicense.com/no-license,* Apr. 6, 2016