CS CAPSTONE TECHNOLOGY REVIEW

JUNE 12, 2018

OSU ROBOTICS CLUB MARS ROVER GROUND STATION

PREPARED FOR

OSU ROBOTICS CLUB

NICK MCCOMB

PREPARED BY

GROUP 30

GROUND STATION SOFTWARE TEAM

CHRISTOPHER PHAM

Abstract

This document examines the differences between three technologies in three different subjects. Python GUI frameworks are going to be used on the project to show and abstract information coming from the rover towards the end user. The arm and graphical visualizers will be used to show how an object or the rover's state. Mapping software is going to be needed for any autonomy or any view of dangers on the map. This document goes over these different subjects and reviews technologies that will be suited for the system.

CONTENTS

1 INTRODUCTION

My section of the technology review document is going to revolve around the ”graphical” side of the ground station software.

2 PYTHON GUI FRAMEWORKS

2.1 Overview

A graphical user interface (GUI) is used by a program to obscure and allow the user to access and interact with objects instead of a command line interface system. The GUI allows for people to interact with the system even if they are not trained. A GUI framework is a pack of tools that will build a GUI using a commands and bindings and will allow for bindings across many systems like Linux, OS X, Windows, Android, and iOS. A framework also comes with an added benefit of being made and maintained by the public or a company instead of being made in-house.

2.2 Criteria

Given the restrictions at the request of our client, I've chosen to focus more towards Python based implementations of graphical user interfaces. Some potential choices can be used in other languages but that is not a main focus point of this project. The interface needs to be able to go into full-screen and needs to be able to change the colors of the GUI itself to a darker theme. It also needs to support many video streams.

2.3 Potential Choices

2.3.1 PyQT

PyQt is a set of bindings for Python for the GUI framework Qt developed by The Qt company. [1] PyQt can run on any platform that can support Qt like Windows, OS X, Linux, iOS, and Android. Qt is not just a GUI toolkit, but also comes with ”abstractions of network sockets, threads, Unicode, regular expressions, SQL databases, SVG, OpenGL, XML, a fully functional web browser, a help system, a multimedia framework, as well as a rich collection of GUI widgets.” [1] Qt also includes a designer called Qt Designer and PyQt is able to generate Python code from it.

2.3.2 Tkinter

Tkinker is GUI framework that is included in any Python 2 or 3 installation package. Tkinker itself is a wrapper on top of the Tcl/Tk package from ActiveState. [2] Tkinker supports ”most Unix platforms, as well as on Windows systems”. [2] Tcl/Tk comes with a BSD-like license that allows for commercial use of their framework.

2.3.3 Kivy

Kivy is an open source framework for Python that is under MIT license. [3] Kivy runs on Linux, Windows, OS X, Android, iOS and can use the same code for all platforms. It can natively use most inputs and devices including ”WM_Touch, WM_Pen, Mac OS X Trackpad and Magic Mouse, Mtdev, Linux Kernel HID, TUIO”. [3] Kivy also includes hardware acceleration using OpenGL ES 2.

2.4 Comparison

With all these frameworks, Tkinker/(Tcl/Tk) and PyQt/(Qt) are also possible on other languages using some other wrappers, unlike Kivy. However, unlike the other frameworks, which are built in C or C++, Kivy is natively built in Python. All the frameworks support OpenGL contexts (windows/frames) only PyQt and Kivy support OpenGL acceleration on the menus which might be necessary for how many video streams we are pushing. Kivy in contrast to the other frameworks also supports multi-touch enabled devices like phones, and tablets unlike the other frameworks. Tkinker however is included in all builds of Python released, making it very easy for quick and lightweight programming and prototyping compared to other frameworks. Qt requires payment for any commercial product, unlike the others.

2.5 Decision

My suggestion for this project would be PyQt. PyQt's fee that Qt asks for is not applied to this project. PyQt and Qt in general have good support and documentation because its longevity, numerous tutorials, and support by the community. Kivy would work well for the project if it had more documentation but its OpenGL rendering would be fantastic for design ideas we have. Kivy is also built more towards multi-input systems which is not our focus. Tkinker would work well for small projects but the lack of OpenGL acceleration might be questionable for the all the video streams that might be handled on the system.

3 Arm/Graphical Visualization

3.1 Overview

For this project, one important aspect is going to be visualizing how the arm is currently moving in respect to the rover itself and showing how the user input changes the arm visually. The need comes from the integration between our project and the other senior project that revolves around the rover arm. Without the visualization, the other senior project can not test to see if their arm works correctly on the rover and for later competition usage.

3.2 Criteria

If this was to be made using the command line or via text, it would be very confusing. Per requests, the arm visualizer needs to resemble the arms. The other requirements would be needing to able to run on Linux (Ubuntu) and easy integration with the ROS (Robot Operating System) on the Rover.

3.3 Potential Choices

3.3.1 ROS Visualization

Inbuilt into ROS, there is a package called rvis which allows for the system to visualize objects using displays. The included displays in rvis are [4]:

Axes	Effort	Camera	Grid
Grid Cells	Image	Interactive Marker	Laser Scan
Map	Markers	Path	Point
Pose	Pose Array	Point Cloud (two types)	Polygon
Odometry	Range	RobotModel	TF
	Wrench	Oculus	

The package is also very well documented with sample code and projects to show how some interactions happen.

3.3.2 OpenGL visualization

We can build a system using one of our GUI frameworks to give us access to an OpenGL window. In that window, we can then write to it using some OpenGL and Python code to form the arm. We can use any shapes we want like line pieces or even importing the arm's 3D model and modifying it from there. Inverse kinematics is needed to get the joints working correctly if the arm is moving from point to point, and normal kinematics to get the final location from arm segment movements. We would be able to control how fast the video/render would be refreshing, drawing, or rendering size which would allow for granularity control to reduce CPU or GPU usage.

3.3.3 Moveit Simulator

We could also use simulator software that can take the signals from the inputs/joysticks and then send them both to the rover and simulator. The simulator can emulate what the arm SHOULD do and can be imported from the other senior project that is already simulating the arm using the same software. The video can be captured using OpenCV or open source prorgrams like Istanbul to stream the video to the GUI.

3.4 Comparison

Compared to ROS and OpenGL, Moveit actually just emulates the arm while the others calculate how the arm should move using inverse kinematics. ROS has all built-in displays in which many other things like mapping software and waypoint systems would be much simpler than building our own system for the project. OpenGL provides flexibility compared to the others but does not include any built-in packages or anything of the sort.

3.5 Decision

I think that the visualization should be a mix of ROS and some custom OpenGL. The combination of the two allow for access to packages that are included and built around ROS while also including the flexibility that is included with OpenGL. The arm can be visualized using the included packages like rvis, but other things like the IMU (Inertia Measurement Unit) might be hard to visualize in rvis and would be possible with custom OpenGL code.

4 MAPPING SOFTWARE

4.1 Overview

Another part of the project is the mapping system. In competition, the Robotics team wishes to see where the rover is currently on the map. We are going off the assumption that the way-point system needs to be built, but we can use a package to deal with the GPS location and mapping.

4.2 Criteria

A requirement would be ease of integration between the rover and ground station. Another would be the ability to use the mapping software with a reasonable amount of delay or no Internet at all.

4.3 Potential Choice

4.3.1 ROS Packages

There are inbuilt packages for GPS and GPS location parsing in the ROS subsystem. Using packages like rvis or mapviZ can display the map and transform the map image to display the correct location. The location can be calculated on the rover or on the ground control system.

4.3.2 Google Maps/Earth

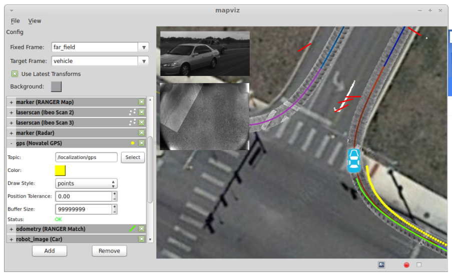
Using the same packages as above, instead of parsing the location on the robot or ground control system, the rover/ground station can parse the information given to it by sending it to Google. This would remove any need for calculating location at all and the map would be generated by Google.

4.3.3 Custom Mapping/Packages

We could build a system using our own polling and parsing of the GPS information. This can be represented by texture location using an external package like Kartograph [6] for Python. Kartograph can then return an SVG (Scalable Vector Graphics) which can be displayed on the GUI.

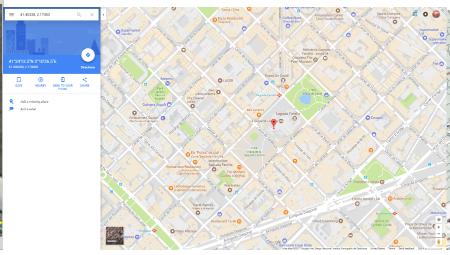
4.4 Comparison

ROS and Google are very similar with the only differentiating thing about the two would be where the information being sent, locally or via ground control to a remote server. ROS and the custom software are very similar but uses in-house code or open source code. The only difference between all of these is how the GPS data is analyzed and texture map is generated and then put onto the screen.

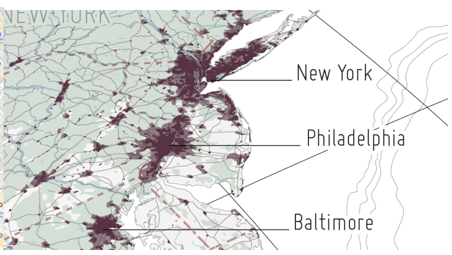


Fixed Frame: far_field
Target Frame: vehicle
Use Latest Transforms
Topic: /localization/gps
Draw Style: points
Position Tolerance: 0.00
Buffer Size: 99999999
Status: OK

(a) Map Visualization Using Mapviz Package [5]



(b) Google Maps using Decimal Degrees



(c) Kartograph Render [6]

Fig. 1: Map Visualization

4.5 Decision

I think that the project should use the ROS packages because all of those are open source and packages are constantly made for problems like this. ROS also has integrations between other packages like rvis and mapping software. And in the environment where the competition is going to take place, the speed and the Internet quality might be sub-par and any external requests would make the system react very slowly or be completely off. Another problem with using Google would be that we can not use their mapping system to do anything autonomous as stated from their ToS (Terms of Service) and we would need to use some other system like OpenSourceMaps. If we had to build a custom solution to this project, Kartograph would be good. Kartograph produces SVG files which Qt takes for but the lack of documentation might make building the project longer than alloted.

References

[1] ”What is PyQt.” Riverbank Computing Limited. November 2017

[2] ”Tkinker – Python interface to Tcl/TK” Python Software Foundation. November 2017

[3] ”Kivy.” Kivy Organization. November 2017

[4] ”rvis/DisplayTypes” David Gossow. August 17, 2013

[5] ”Mapviz.” swri-robotics, November 21, 2017

[6] ”Kartograph - A Simple and lightweight frameowkr for createing interavtive vector maps”