

The University of Alabama in Huntsville

Autonomous Mortar Remote Control Center

Honors Capstone Project

James Tyler Frame

A handwritten signature in black ink, appearing to read "J. Tyler Frame", written in a cursive style.

5-4-2021

Table of Contents

| | |
|-------------------------------|----|
| Abstract..... | 2 |
| Section 2: Environment..... | 2 |
| Section 3: The Queue..... | 2 |
| Section 4: The Map View | 3 |
| Section 5: The Chart..... | 4 |
| Section 6: File IO | 6 |
| Section 7: Connect | 7 |
| Section 8: Menus | 7 |
| Conclusion..... | 8 |
| Reference List..... | 10 |

Abstract

The objective of the Autonomous Mortar Remote Control Center, or AMRCC, is to be an add-on of the Autonomous Mortar senior design project giving more flexibility and informational capabilities to the user. The Autonomous Mortar project is focused on developing an unmanned mortar system focused on automatically firing smoke rounds to provide fast cover for members of our Armed Forces that are ambushed in areas with little cover. The main purpose of the AMRCC would be to assume control of the Autonomous Mortar in a remote and secure manner. Its additional capabilities include a map giving a visual display of the Autonomous Mortar and its targets, a 2D chart plotting the trajectory while displaying other known and calculated values of each shot, and file import/export with information for each shot.

Section 2: Environment

For the actual development of the AMRCC, Qt Creator was used for its quick and simple user interface design as it seems to be the biggest time sink of anything I've developed while attending UAH. While studying at UAH, I had the opportunity to take a course focused on learning Qt so I was familiar with it beforehand which also helped me make my decision in choosing it for my development environment. Qt allowed me to see visual changes as I made them and determine whether or not it was a change I actually wanted to make. The signal and slot system also gave a more convenient route for calling functions when needed, not requiring me to maintain references to different objects I wanted to interact with.

Section 3: The Queue

As the AMRCC was developed there became a clear need for a central interactable location that could be drawn on for information to help support all of the other capabilities. This queue of shots is displayed on the left side of the window at all times. It has most capabilities expected of a queue including resizing, automatic removal of old shots when the queue has reached max size, locking important shots to avoid automatic deletion, etc. As can be seen from the figure to the right, each index of the queue has four primary functions available to the user through QPushButtons and QRadioButtons. The QPushButton on the top left allows the user to set the index as the AMRCC's selected shot. When this happens, the index is drawn on for information to populate the 2D chart capability as well as being highlighted in red on the map. The QPushButton on the top right opens a modified QDialog with the capability of allowing the user to see read-only information and also manually set other information such as impact distance from target as well as the compass reading from the target to the landing location. These two editable values become important to the user when attempting to keep track of actual data related to the Autonomous Mortar's accuracy and precision. The last two interactable objects on each index are a manual deletion QPushButton and locking QRadioButton which disables automatic deletion when the queue is in need of space.

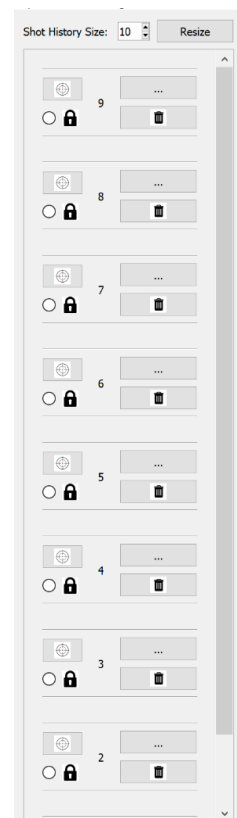
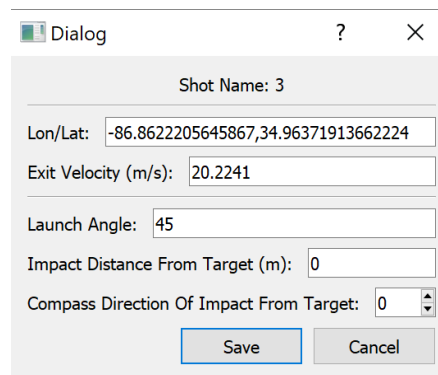


Figure 1. The queue populated with shots.



Dialog

Shot Name: 3

Lon/Lat: -86.8622205645867,34.96371913662224

Exit Velocity (m/s): 20.2241

Launch Angle: 45

Impact Distance From Target (m): 0

Compass Direction Of Impact From Target: 0

Save Cancel

Figure 2. The custom QDialog allowing the user to see read-only values as well as editable ones for each shot index in the queue. The values between the save button and the next horizontal spacer from the bottom are the editable values.

Section 4: The Map View

The map view for the AMRCC was something I was nervous about completing since it relied on getting up to date maps with custom markers, and this wasn't something I had done before. I was very relieved when I came across a company named Mapbox. Mapbox has Software Development Kits and well documented Application Programming Interfaces (APIs) available, some of which were available to users for free. I chose to use their API for retrieving static images because there wasn't a need for the AMRCC to have a live display of the map if changes only occurred when adding or deleting a shot or setting a shot as the new selected shot. Mapbox also had many settings for their static image API I could make available to the user through a modified QDialog for editing those settings.



Figure 3. The map with the Autonomous Mortar represented by the T marker while each of the numbered markers is a target. The target numbered three is highlighted red because it is currently the selected shot.

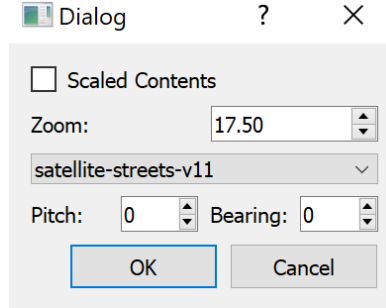


Figure 4. The modified QDialog giving the user access to settings of the map.

The only setting that might not make sense from the figure above is the scaled contents QToggleBox. It is for stretching the retrieved image to fit the area of the QTabWidget it sits inside if the user wants it to be that way.

Section 5: The Chart

As I got around to the chart, I made use of formulas gathered by a member of the Autonomous Mortar senior design team. This resulted in a rather quick implementation as I could just use his source as reference right away. In the following equations, numerous variables are static values for these calculations: m is mass of the projectile, g is the force of gravity, v is the initial velocity, n is the launch angle in radians, A is the surface area of the projectile from both x and y facing sides, C is the drag coefficient of the projectile, and p is the average density of air.

$$k = \frac{1}{2}pAC$$

Equation 1. Inertial drag force coefficient of the projectile.

$$\{0 < t < b_1; \frac{m}{k} \ln \left(\cos \left(t \sqrt{\frac{gk}{m}} - \arctan \left(v \sin(n) \sqrt{\frac{k}{mg}} \right) \right) \right) + \frac{m}{2k} \ln \left(\frac{kv^2}{mg} \sin^2 n + 1 \right) \}$$

Equation 2. Path of the projectile from launch point to the apex.

$$\{b_1 < t < b_2; \frac{-m}{k} \ln \left(\cosh \left(t \sqrt{\frac{gk}{m}} - \arctan \left(v \sin(n) \sqrt{\frac{k}{mg}} \right) \right) \right) + \frac{m}{2k} \ln \left(\frac{kv^2}{mg} \sin^2 n + 1 \right) \}$$

Equation 3. Path of the projectile from the apex to the landing point.

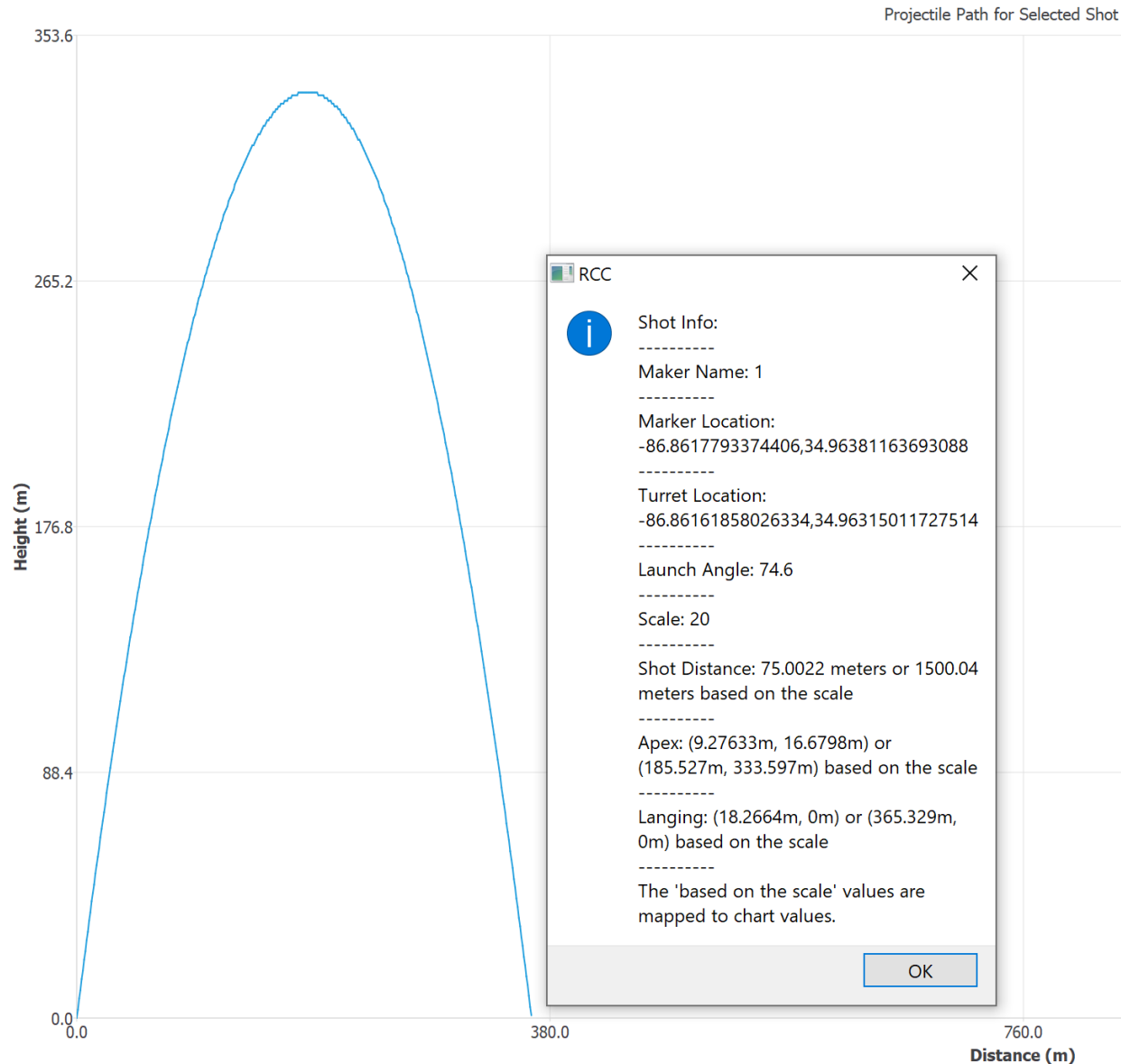


Figure 5. The resulting projectile path based on two given locations and a launch angle.

In the figure above, the x and y axis are on scales much larger than the expected values. This is because the QtChart object I was using to plot the line didn't want to plot points on non-integer values. This resulted in very blocky paths that did not visually represent the true path well. To account for this I simply scaled up the chart so I had the needed number of points to make the path look smoother. The user can even adjust the scale in the modified QDialog displaying the chart settings. If the scale is not a value of one then the chart's numbers won't line up with the correct values. In order to communicate this to the user I have a popup dialog show general information of the projectile path if the user wants it. This popup also maps different points from their true values to values the chart is displaying if the scaling is greater than one.

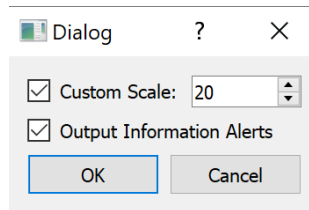


Figure 6. The custom QDialog the user can make changes to the chart settings from. In this case the user has chosen to use custom scaling of 20, and also have information alerts output after the line has been generated.

Section 6: File IO

For the AMRCC, another main capability it has is file input and output. This can be handy when exporting data to be used in other applications or when being used by other programs to gather data from the information. I used comma separated value (.csv) files because these are very simple files information can be read from and written to. There is an import file QPushButton with the purpose of opening a QFileDialog at the standard saved file location inside the AMRCC's build folder where users can choose from pre-existing .csv files to import. When it comes to the save function, there are more options for how it is done. The standard save filename is based off the date and time the application was started up or reset. The user can also choose to append to a pre-existing file if desired. There are three different save modes, one being a combination of the other two. The save queue option will only save the current turret location and shot information existing in the queue. The save input option will write to the save file as updates come in for either the turret or target locations. The save both option will perform a queue save and then carry out input saves for the rest of the time the save is activated.

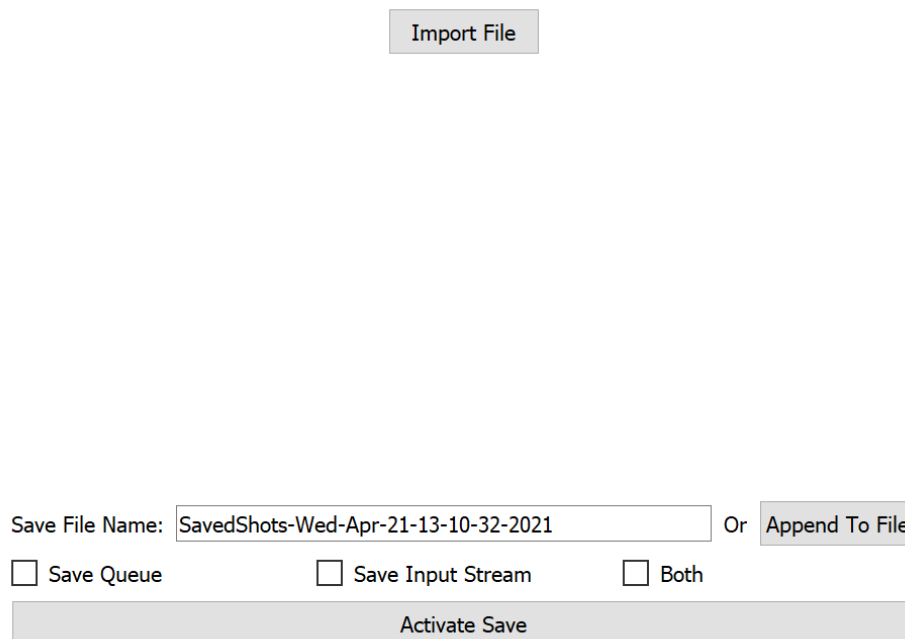


Figure 7. Image of the file IO options.

Section 7: Connect

In order to allow the user to assume control of the Autonomous Mortar, there had to be a way to communicate with it. This was accomplished by taking a discord bot template developed to allow the Autonomous Mortar communicate with its marker. After making modifications to the template, the AMRCC had its own discord bot with all the base commands of the other bots as well as role specific commands. The connect tab of the AMRCC is rather plain. There is a connect QPushButton for starting the discord bot. When the connect button is pressed, it takes a discord bot token from the QLineEdit to the right of it and attempts to sign in. Below these two is a read only QTextEdit which has messages posted to show the communication of the AMRCC's bot and the other bots.

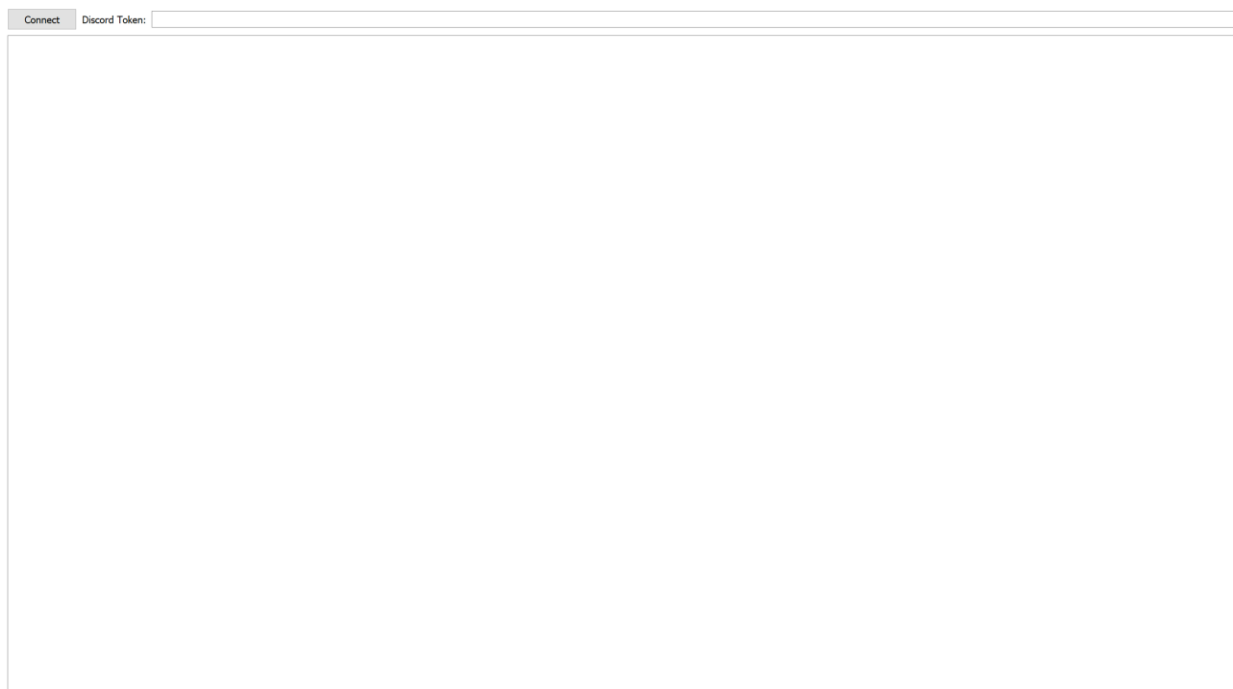


Figure 8. The AMRCC's connect tab.

Section 8: Menus

For calling numerous functions, I used the menu and action system of Qt to build out Operations and Settings menus. The Operations menu holds actions that will trigger the AMRCC's discord bot to request for the turret and marker's global positioning system, or GPS, locations for purposes of populating the queue, map, and chart. There are also manual routes for submitting GPS locations for each of these objects under this menu. The Operations menu also holds actions that have the AMRCC's discord bot request for either the turret or marker for their satellite count, which is used when the location cannot be found and the user wants to see if their GPS module has view of any satellites at all. The next to last action in this menu is the Fire At action which triggers the AMRCC's discord bot to tell the turret to fire from its GPS location to another GPS location. The final action is directed toward resetting the AMRCC to a state that was the same as if it had just started up. Under the Settings menu there are only two actions. One of them opens the map setting's custom QDialog while the other performs the same action for the chart setting's custom QDialog.

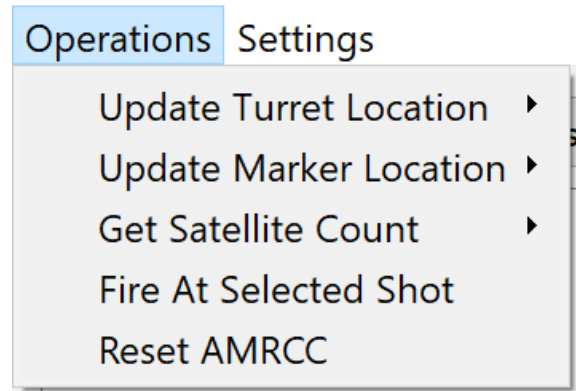


Figure 9. The Operations menu and all of its actions.

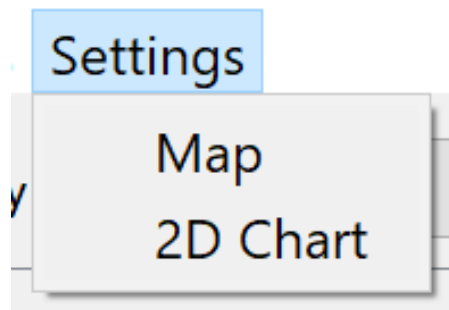


Figure 10. The Settings menu and its actions.

Conclusion

With this project's development coming to an end, I can say I am very proud of this application. It is the most polished thing I've developed, and it is very satisfying to not be immensely rushed to complete such a task. I will admit there are still issues with the AMRCC. The discord bot used from the Autonomous Mortar isn't the best design. It does require the user to have the correct token and use the right script with the needed communication format. However, it doesn't do any confirmation of someone being who they say they are. If I could go back and rework it, I would probably look for some type of encrypted email system with an API, then plain text messages could be sent and let the email service worry about encryption/decryption, confirmation of the person, etc. These issues focus more on the Autonomous Mortar project's design than the AMRCC's, even though they are very closely related. There is also a high possibility I wasn't able to think of or encounter all the use cases of different user/application interaction that might cause it to crash. I also would have liked to make this its own release build so it could be passed around easier without needing Qt Creator to run it, but the class on Qt at UAH didn't cover the standalone build aspect. I did it for a much smaller side project, and I was unable to find an easy way to do it since Qt didn't build the release with all of the needed Dynamic-Link Library, or .dll, files. For the smaller project, I ended up manually digging through the Qt files and copying the needed .dll files, but this project is much larger and would require a larger amount of .dll file tracking which was very frustrating for even a small project. I may have just not found the right resources in my research for creating a standalone release build or missed a specific setting in Qt

Creator for it, but manual copying was the only route I found worked. I want to continue to working on the AMRCC since it feels unfinished simply because I know it could be better, but most of these changes would require time that is simply not available. The main thing I would change is the communication method, but it is a major part of the Autonomous Mortar project which is coming to a close so it's not possible. As stated earlier, I am still proud of this project even with all its flaws. It has greatly contributed to my experience as a computer engineer in what I believe is a beneficial manner.

Reference List

Frame, James. “Autonomous Mortar Remote Control Center Code Repository.” *GitHub*, 21 Apr. 2021, <https://github.com/Ty-Frame/AMRCC>.

Frame, James. *AMRCC Demo*. *YouTube*, 23 Apr. 2021, <https://www.youtube.com/watch?v=FmprHxttjL4>.

“Projectile of a Trajectory: With and Without Drag.” *Desmos*, <https://www.desmos.com/calculator/on4xzwtdwz>.

“Static Images: API.” *Mapbox*, <https://docs.mapbox.com/api/maps/static-images/#overlay-options>.

Title: Autonomous Mortar Remote

by

Name: James Tyler Frame

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma ☐

to


The Honors College

of

The University of Alabama in Huntsville

Honors Capstone Director: Dr. Earl Wells

Program Director: Dr. Ravi Gorur

 5/4/21
Student (signature) Date

 4/21/2021
Director (signature) Date

Department Chair (signature) Date

Honors College Dean (signature) Date



Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the Capstone manuscript.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Ty Frame

Student Name (printed)

Ty Frame

Student Signature

5-4-11

Date