Expanding Virtour

Walter Sagehorn

***Abstract*—Virtour, developed by the Building Wide Intelligence Laboratory at UT Austin, is designed to give web-based virtual tours of the Gates-Dell Complex at UT. This project improves Virtour by making it more user friendly and informative as a tour system and improving its capability to be utilized as a research tool. This was accomplished by integrating CMASS (Centralized Multi-Agent Status Server) into Virtour as a scalable and secure means of data transfer. The inclusion of CMASS into the BWI system allowed improvements to be made on the user-facing end of Virtour and also introduces a tool for robot status dissemination across the entire lab system. This paper also explores other current issues in Virtour and possible future work.**

I. Introduction

Virtour has proved to be a benefit to the University of Texas as a platform for virtual tours and the Building Wide Intelligence (BWI) lab as a research tool. However, there are some areas in which Virtour can be improved. Virtour, despite being a virtual tour, does not provide those on tours with information about the things they are seeing or what the robot is doing. Another area that Virtour could be improved in is shared autonomy among users. Currently, one user acts as the ”tour leader” and provides the robot with high-level objectives. As suggested in previous studies, it would be restrictive to the majority of the users if only one controlled the experience.

CMASS was originally developed to replace the existing DNS with a system that could keep track of more than just an IP address. It was soon realized that this could also prevent direct communication between the robot and Virtour application, which could cause problems for both parties.

II. Related Work

A. Virtual Tours

The concept of using robots to give virtual tours is not new; much of the early research was done on robots giving tours of museums. Some of the first work was done by the University of Bonn in Bonn, Germany. A team of researchers built an autonomous mobile robot called RHINO[3][4] that moved around the museum and interacted with visitors in person and on-line through a web interface.[2] The web interface[9] was a Java applet like the one in fig. 1) that gave background information about the robot along with photos from the robot's cameras obtained using the client-pull technique. The web interface also gave users the ability to direct the robot through high-level objective commands. The researchers experimented with differing levels of shared autonomy and shared control between on-line tourists and tourists physically on location. They stressed the importance of low-bandwidth communication and placing all relevant robot information on one web page. They also cautioned that giving a single user total control may restrict access to the robot to a unacceptable degree. They later revised the robot and web page (seen in Fig. 2)) and deployed it at the Smithsonian museum.[10]

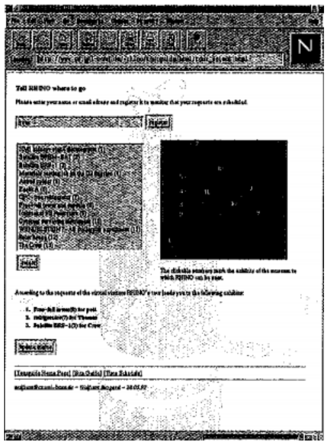
Fig. 1. The Java applet used to interface with RHINO.

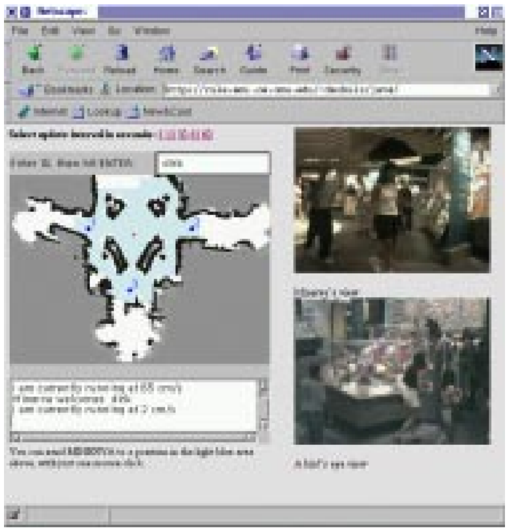
Fig. 2. The update Java applet used to interface with Minerva.

These studies give tours of sites by visiting their location, but it is important to not just visit a location but to prominently display the object of focus for the on-line tourist to view. Researchers at the National Chengchi University in Taiwan developed a path-finding algorithm that takes what an agent sees into account, which would be very useful for solving this problem with tours.[6] Virtour, the project this one seeks to improve upon, is also a system for virtual tours. It features live video feed, a single tour leader, and high level directives like ”visit room.” [5]

B. *Multi-robot Communication and Security*

Authors of an editorial on the state of multi-robot systems outlined the seven principle topic areas as:

- Biological Inspirations;
- Communication;
- Architectures, task allocation, and control;
- Localization, mapping, and exploration;
- Object transport and manipulation;
- Motion coordination; and
- Reconfigurable robots [1]

CMASS would most likely fall under ”Communication,” as it serves primarily as a method of communication between robots and a server. A central concern of CMASS is security- a broad field of research in itself. Robert Morris and Ken Thompson provided a case history on passwords and using them to secure a system.[8] A more niche element of CMASS is key stretching, which is described in detail by Colin Percival in his paper about key derivation. [7]

III. Methodology

A. *Building a centralized multi-agent status server*

Before the Virtour system could take advantage of CMASS, it had to be implemented and integrated into the robot system. As stated before, CMASS began as a replacement for the DNS that mapped robot names to IP addresses. The DNS, called smallDNS, was originally written in Python. It consisted of two parts, a program running on the server that could distribute information as a web server and also contact robots to determine if they were running. The second part was a script that ran on the robots that periodically pinged checked for an IP address change and, if it found its IP had changed, it would ping the server to update its entry. This method of direct-robot communication isn’t scalable in either direction; Virtour with many robots would be unusably slow while many simultaneous Virtour connections to a single robot would essentially DDoS it and prevent it from performing higher priority tasks like navigation. Also, since the server was written in Python, it lacked efficient concurrency. The server was rewritten in the Go programming language to solve this problem. This rewrite became the basis for CMASS, which is also written in Golang. Once it was clear that useful information could be sent in the ping to the server, we began to think of ways to send it. It made sense to send them via HTTP requests because the server can then be a single HTTP handling program and handle robot traffic and web traffic from Virtour or other places. The cURL C++ library was used to send the request from the robot. Since the valuable information from the robot was available in the form of ROS topics, it was reasonable to write the client program as a ROS node that could subscribe to topics and send their contents to the server via cURL. Since the ROS node would only run when the BWI robot code was running, there needed to be a way to kept the robot’s IP address up to date when the robot is on but not running ROS nodes. An approach similar to the one taken by smallDNS was taken- a python script that checks for an IP change and pings the server if it finds one was written and runs periodically as a cron job. The server is essentially a standard web server that takes HTTP requests from two different groups and as such it has 2 types of URLs. The first type serves to update entries and consists of one path, /update. Robots pass data to the server as URL parameters after the /update path. The server responds with affirmation that the update was successful, unless there was an error, in which case it returns an error message and does not update the record of the robot. The other type of request is from the web, usually Virtour. These paths provide access to the server’s stored robot information. Below are the supported paths and what they serve:

/text	all robot info in human-readable format
/json	all robot info in JSON
/hosts	all robot name:IP pairs in text*
/hostsjson	all robot name:IP pairs in JSON*
/hostsalive	active robot name:IP pairs in text*
/hostsalivejson	active robot name:IP pairs in JSON*

* - Indicates inclusion for legacy support. The system also automatically saves a copy of the data to disk when a record is updated. When the program is started, it looks for a local copy of the previous run’s data so it can pick up where it left off. If it doesn’t find one, a new one is made when the first update is performed.

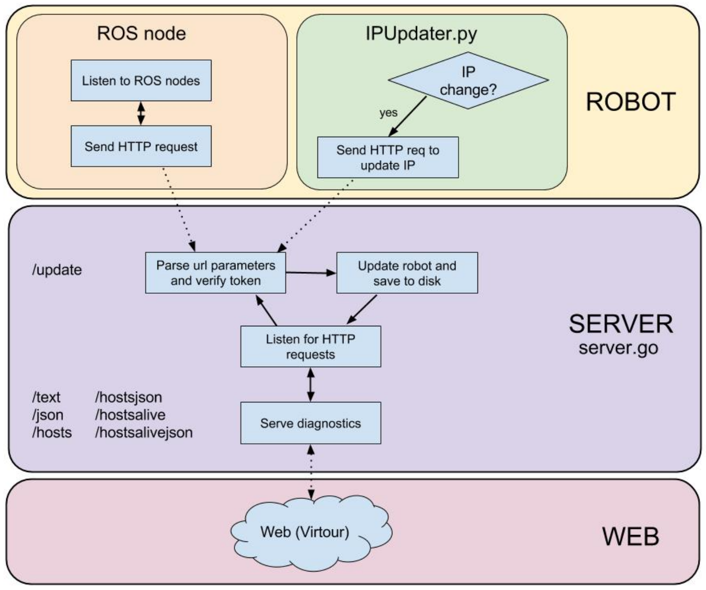

Fig. 3. The structure of CMASS.

B. *Securing CMASS*

Since the server that CMASS is running on is accessible from outside the UT network, there needed to be some way to secure the system. Ideally, we would like to be able to confirm that the update request is coming from a robot within

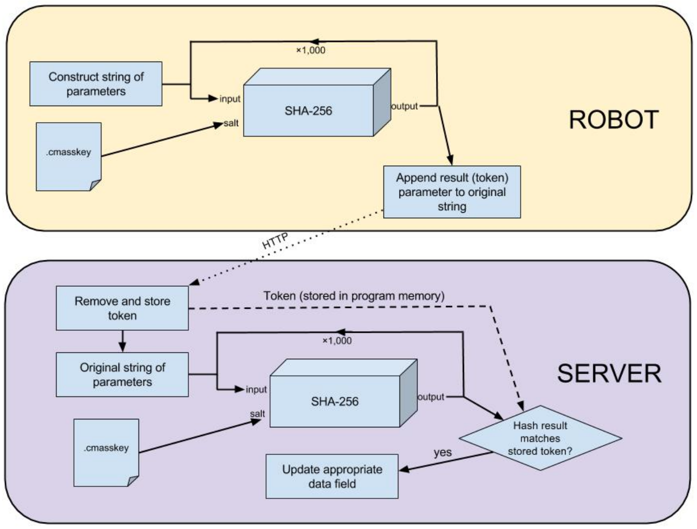
ROBOT
Construct string of parameters
.cmasskey
×1,000
input
salt
SHA-256
output
Append result (token) parameter to original string
HTTP
SERVER
Remove and store token
Token (stored in program memory)
Original string of parameters
×1,000
input
salt
SHA-256
output
.cmasskey
Hash result matches stored token?
yes
Update appropriate data field

Fig. 4. The flow of data within the CMASS security system.

the lab. The first solution that was proposed was to just leave the system unsecured and trust that attackers just don't find the system or deem it worthy of being tampered with. Another idea proposed was to use SSH to modify the local copy of the robot's statuses.

The solution that was implemented is a token based authorization system. It depends on a key known by both the client and server beforehand. When a client wants to prove that a message is being sent by a trusted source, it hashes the message using the key as salt. The hash function used was SHA-256. It feeds the result back into the same function again, again using the key as a salt in a effort to avoid reducing hash space by repeated hashes on digest. This process is repeated until the hash function has been performed 1,000 times; the resulting digest is the token. It is repeated so many times because of a technique called "key stretching." If you think of the entire 1,000 hash process as a single process, it is much more expensive to perform than a single hash. Because all 1,000 are needed to compute a token, it is much more expensive for an attacker to brute force a collision.

The token, along with the original diagnostic information, is sent in the form of URL parameters to the /update path. Upon receiving a request, the server removes the token from the parameters and stores it for later. It then performs the same token computing process as described above and compares its result to the token it received from the robot. This process is seen in Fig. 4. If they do not match, the request is disregarded; if they do, the server moves to the next criterion. Since the parameters contain a timestamp, they are time sensitive. If this was not the case, an attacker could intercept a valid /update request and continuously send it to the server, freeing the robot's state to the one the request described. To prevent this, the server checks the timestamp sent in the parameters- if it exceeds a certain timeout or happens in the future, the request is disregarded. If not, it is a valid update and is applied to the store records.

An unexpected complication was encountered when tokens from a certain machine seemed to always time out. It was determined that the cause of this was that the machine's system clock was substantially different from the server's. To account for this and to prevent similar issues in the future, the timeout window's size was drastically increased.

C. *Live Robot Locations*

Now that up-to-date robot information was readily available, the next step in displaying live robot locations was to write the JavaScript that was going to display them. A lot of time was spent combing through the massive, single-file program that performed almost all of Virtour's functionality to find what things to change. It was decided that the robot locations should be displayed on the home page of the application and replace the current page that just displays tiles of robots who are alive, as seen in fig. 5. The robot locations are displayed by positioning location markers over an image of a map of the lab (fig. 6). When clicked, the a popover extends from the pin and gives the user more information (fig. 7). If the robot is available to stream be viewed for a tour, the "view" option is present (fig. 8).

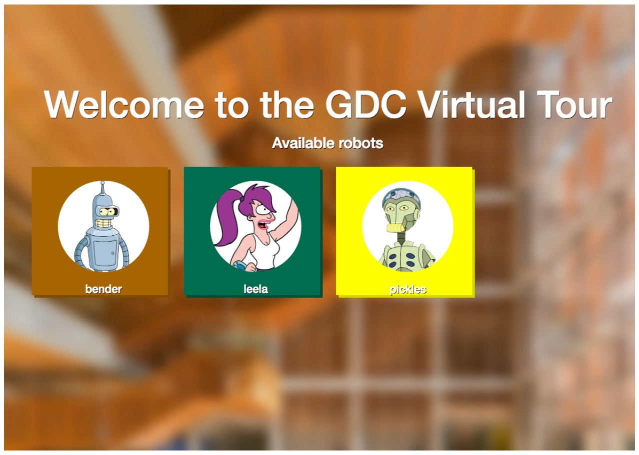

Fig. 5. The old Virtour home page.

Other aesthetic and easy-of use changes we also made to Virtour. A back button was added to the tour page to allow the user to quickly navigate back to the home page. The shadow on the text was also removed and a few colors were tweaked.

IV. Conclusions and Future Work

Hopefully, this project was an important improvement to Virtour and the addition of CMASS will prove to be a valuable asset to the lab.

A. *Future Applications of CMASS*

CMASS could be used not only to communicate with other applications, but also facilitate communication among robots. The communication wouldn't be direct, but robots could learn about each other's positions and directives and adjust their own

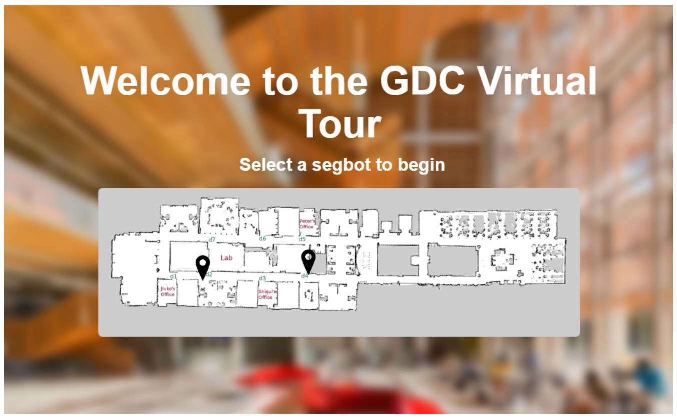

Fig. 6. The updated Virtour home page, with live robot locations.

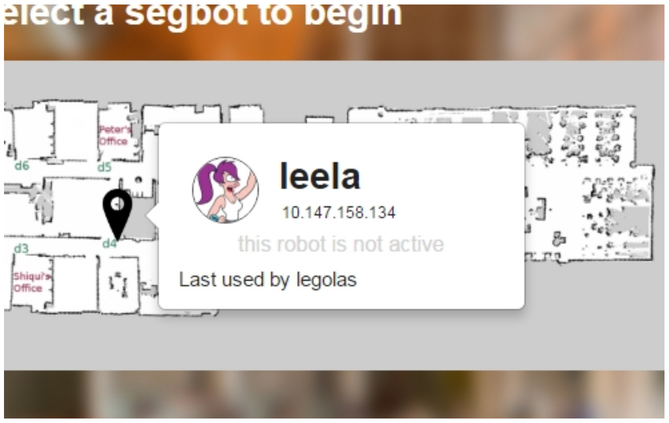

Fig. 7. The information display when a pin is clicked.

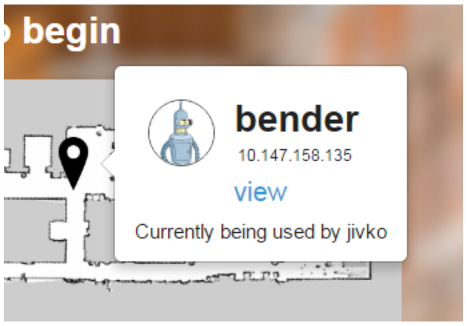

Fig. 8. The view option present when a robot is active.

plans accordingly. CMASS is also highly expandable- almost any ROS topic that can be subscribed to can be sent and stored on the server. The only exception are media files like photos and video streams, which would be inefficient to send in a request like CMASS does.

B. Current problems with Virtour

Virtour has a few problems that aren't exactly features to be added, but more like behind-the-scenes things that could really be improved.

The first is the structure of the code. The code that handles almost all of what Virtour does is all piled into one 1,000 line JavaScript file. It's hard to work with an steepens the learning curve, which may prevent people from working on this project. It should be commented more and there are areas that need to be refactored (my code included).

Another issue is how the video streaming is done. A package called web_video_server is used to stream video from the robot. It's accessible via a direct connection to port 8080. There are two problems with this:

1) This direct connection limits users of Virtour to people within the UT network because access to the robots is restricted.
2) A direct connection with the robot means that every instance of Virtour requires video to be streamed to it. In other words, a robot must stream to every client that's watching it, and that stacks up quickly.

This could be solved by streaming to nixons-head, the server which runs CMASS and is accessible outside the network, which could somehow temporarily store the video and send it to Virtour users who request it.

C. Improvements to Virtour

One of the things I addressed in my proposal but didn't implement in this project is ”more tour-like tours,” which basically means providing relevant information through pop-ups throughout the tour. For instance, fig. 9 shows the current view a user sees all the time when participating in a tour. Fig. 10 shows a crude example of a information pop-up that could display more information about what the tourist is seeing. This would require some manual labor- someone has to write the information and determine the location at which the robot would display it.

Another major improvement to Virtour would be multi-floor capabilities. This would allow tours to span floors and users to see much more of what the GDC is like. The mapping system implemented in this project would have to be expanded the accommodate more maps. Perhaps there could be arrows that allow the user to traverse the different floors. Something that would help with this objective is dynamic map fetching from the robot. The maps would constantly be up to date.

Virtour could also be improved by adjusting the control scheme from a dictatorship-like system where one person determines all actions to a more democratic one in which people vote and the robot goes to the most popular location. Something to consider with this implementation is the possible domination of one or a few destinations. There could be less of a chance to see things that have been recently seen. This seems like it would be a conflict between users of different lengths. Short-term users would just vote for what they want to see immediately while longer-term tourists would get bored of visiting the same couple places over and over.

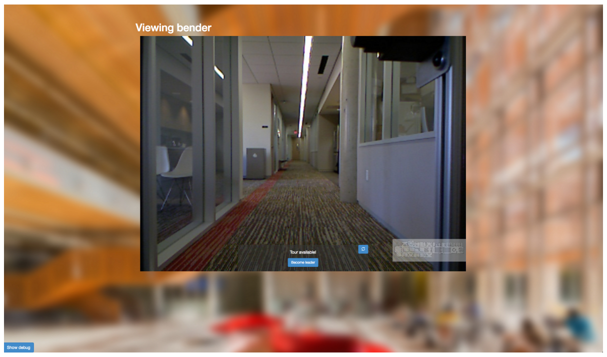

Fig. 9. The current Virtour user's view.

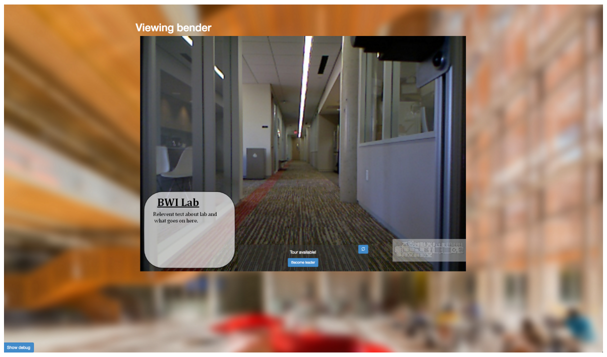

Fig. 10. An example of a pop-up that shows relevant tour information.

References

[1] T. Arai, E. Pagello, and L.E. Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on Robotics and Automation*, 18(5):655–661, 2002.

[2] W. Burgard, A.B. Cremers, D. Fox, D Hhnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Elsevier*, 15.

[3] W. Burgard, A.B. Cremers, D. Fox, D Hhnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. *AAAI-98 Proceedings*.

[4] W. Burgard, A.B. Cremers, D. Fox, D Hhnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The museum tour-guide robot rhino. *Autonome Mobile Systeme 1998*, 15.

[5] Patricio Lankenau. Virtour: Telepresence system for remotely operated building tours.

[6] Tsai-Yen Li, Jyh-Ming Lien, Shih-Yen Chiu, and Tzong-Hann Yu. Automatically generating virtual guided tours. *Computer Animation*.

[7] Colin Percival. Stronger key derivation via sequential memory-hard functions. *BDSCan*.

[8] Morris R. and Thompson K. Password security: A case history. *Bell Laboratories*.

[9] D. Schulz, W. Burgard, D. Fox, S. Thrun, and A.B. Cremers. Web interfaces for mobile robotics in public places. *IEEE Robotics and Automation Magazine*, 49.

[10] S. Thrun, M Bennewitz, W Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: a second-generation museum tour-guide robot. *Robotics and Automation*, 4.