Simulated Robotic Arm Teleoperation using Predictive Display

Rudraksh Kapil and Raghav Madan

Introduction

Teleoperation tasks in remote environments, such as in mines, quarries, or in outer space, require a robot in a possibly unfamiliar environment to be controlled by a distant operator. Due to the large distance between the operator and the robot, there is often a time delay in communication between the two sites (Jin et al., 2019). The unfamiliar environment and communication time lag contribute to unreliable robot control. Latencies of even a few tenths of a second can impair operators. This is where predictive display comes into play. Predictive display gives the operator of a remotely-operated robot a predicted view of the remote site without any delay. To put it another way, the latest delayed image at the operator site is warped to look like the live view based on the live movement command and camera pose information. This allows the operator to move the robot more accurately in the remote environment and complete the task more quickly. Predictive display can be applied in many scenarios where teleoperation is required and latency is an issue, such as in the case of unmanned spacecrafts.

In this project, we simulate a predictive display system, wherein different views of a remote scene are predicted by taking images from a camera mounted on a robotic arm. The goal is to alleviate the delay experienced by the operator in a simple teleoperation task. We opted for a simulation because we can have a communication delay without having the remote and operator sites actually separated by a large distance. Simulations also provide a cost-effective way to compare different implementations of predictive display in a timely manner, which is important considering the timeframe of this project. To make the predicted view more realistic, we project it onto a 3D reconstruction of the scene, so that the relative position of objects at the remote site is portrayed more clearly.

Figure 1 summarizes our teleoperation setup. The operator sends robot commands to the remote site, the robot performs these, and the camera moves as a result. In order to compute the predictive display on the operator site, we need live information about the camera movement. Fortunately, the camera pose information is just a few numbers and can be transmitted much more quickly than larger chunks of information, such as the images from the camera itself. We update the 3D reconstruction of the remote site and

project our predicted view on to the reconstruction. The operator then uses this view to determine the next input, and the cycle is repeated until the task is completed.

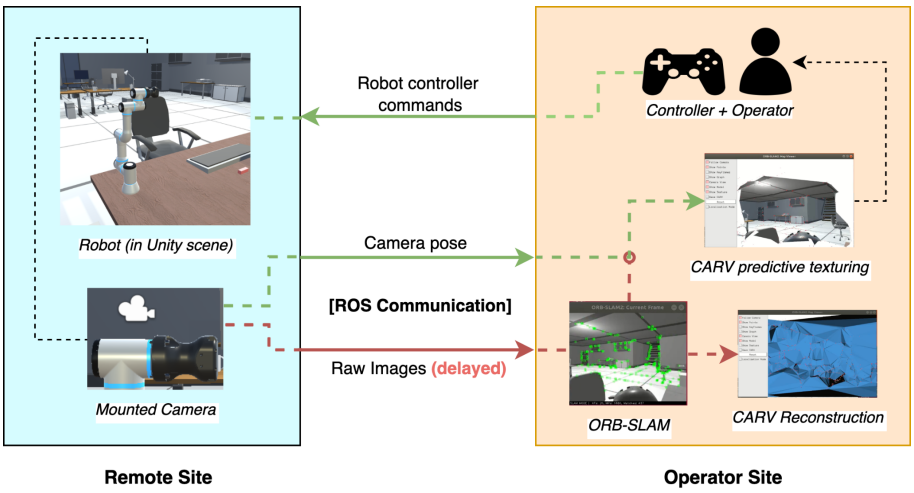

Figure 1. Overview of our simulated teleoperation system.

Related work:

Unity robotic arm

The Unity game engine is tailored for robotics simulations as well as for games. Unity has been used by many researchers for different kinds of robotics simulations. With the recent incorporation of the Nvidia PhysX 4 engine into Unity, the physical interactions between objects have become even more robust and realistic. To facilitate research and applications, the developers of Unity have composed a central repository for tools, tutorials, resources, and documentation for robotics simulation. A demo consisting of a controller for a model of the Universal Robotics UR3e robotic arm is available on their repository, and serves as a good starting point for our implementation of the robotic arm. ROS Sharp has recently been made publicly available, allowing for easier integration of ROS components into Unity projects. This is also advantageous for our simulation as it provides a medium for communication between the remote and operator sites.

3D Reconstruction

Many different 3D reconstruction techniques have been developed. Some of these are offline, where the reconstruction is done from multiple images of an object after the sequence is taken. Online techniques perform the reconstruction as the sequence is taken, incrementally updating the reconstruction with each new image. Since our teleoperation setup runs in real-time and the operator needs a live representation of the remote site, we need an online 3D reconstruction system on which to implement our predictive display.

Lovi et al. (2011) developed an online 3D reconstruction system that leverages incremental free space carving algorithms. These algorithms utilize Delaunay triangulation to carve out space to attain a geometric model of the scene. The graphic texture rendering is done online as well. The reconstruction algorithm follows a three-step process to produce a 3D reconstruction of the scene. First, it gathers point features from a frame and produces point clouds using SLAM-like systems like PTAM (Klein and Murray, 2007). These points are part of a keyframe that is stored in the map. Second, when the map is updated due to the camera movement, various events are triggered, such as new keyframe (camera) insertion. The handlers for these events carve the tetrahedrons in the reconstruction to agree with the updates. The key idea is that if a point is visible from a camera, then all tetrahedrons between the camera position and the point need to be marked as free space. For example, when new keyframes are inserted, tetrahedrons are carved out based on the new positions of the tracked points from the point cloud in the 3D scene. In the third and final step, the reconstruction is extracted as the facets between the carved and uncarved tetrahedrons, resulting in a triangulated isosurface mesh. Once a good model of the scene is available, predictive display techniques can be used to render different views of the scene based on input commands.

Another way to obtain the 3D reconstruction is through the work done by He et al. (2018), wherein the authors extract 3D lines from point clouds to reconstruct the 3D scene. Firstly, a semi-dense point cloud is extracted from the scene using ORB-SLAM. Secondly, edge drawing is used to extract edge segments from the point cloud. Thirdly, the 3D line's points are obtained by solving two 2D line-fitting problems using the points' x, y, and depth coordinates. The points are then filtered to obtain the 3D line in that particular keyframe. Lastly, since each keyframe generates its own set of line segments, they must be converged to form a coherent 3D reconstruction. This is done by filtering, wherein each 3D line is checked with every other 3D line against their alignment and angle. When a 3D structure is obtained, it can be overlaid with the graphics texture and used for predictive display.

Predictive Display

Predictive display is the task of predicting the view of a remotely operated robot/camera if it were made to translate in a remote environment. Cobzas and Jagersand (2005) implement this as a two phase process. The first phase, called the bootstrapping phase, involves tracking a series of points over a hundred or so frames to generate a geometric model and graphics texturing for the scene. The second phase, called the tracking and predictive display phase, is the teleoperation phase, where the robot and operator communicate. The robot sends its pose to the operator, and the predictive display

algorithm computes the next view based on the current position, the geometric model generated in the previous phase, and the movement command from the operator.

In Rachmielowski et al. (2010) the authors propose a video based system that produces a coarse graphics visualization in real-time. Such a system can be used for predictive display as well, especially in an online setting. The system utilized here is composed of two components: tracking and visualization. Firstly, in tracking, when a new image arrives from the camera, features are tracked. New features are also initialized if there are no features to detect using the corner detector. A SLAM update too is done in this part, wherein the current camera pose is estimated. If the pose is novel, the image is saved as a keyframe. Secondly, in visualization, the currently visible 3D points are projected onto images closest to the virtual view, triangulated, and projected back again to get a 3D reconstruction. The system also has a keyframe forgetting heuristic, to keep track of memory constraints. Whenever a new frame is added, it is stored either in the program's main memory or in a disk space. When either memory exceeds a given threshold, images are dropped based on their closeness to the current frame and the uniqueness of their pose.

CARV (i.e. Lovi et al. 2011), which is similar to the work done by Rachmielowski et al, too produces a real-time reconstruction of the scene. In CARV however, only the last available texture is used for texturing the 3D reconstructed surface. Furthermore, In Jin et al. (2019), the authors build upon CARV to use it as a long-range teleoperation system to perform a task in Canada with a robot that was operated by a user in Singapore. Thus, we extend the work done by Jin et al (2019) to make CARV do predictive display in a teleoperation setup.

Methodology

In this section we detail the steps we took to implement our project and the reasoning behind them. There are three subsections corresponding to the remote site, the operator site, and communication between the two.

Remote Site – Unity

The reasons mentioned above in the related works section made Unity a natural choice for our simulation environment. The robot arm asset is provided in the Unity Robotics github repository, but we needed to modify it for our project. The most important change was to mount a virtual camera on top of the end-effector, i.e. an eye-in-hand configuration, and to tweak the control speeds of the various joints so that the task could be accomplished using this kind of viewpoint. Further, we scripted the robotic arm to respond to joystick controller inputs so that the movements could be done more

smoothly as opposed to using a keyboard (even though the joystick wasn't completely smooth either). A total of four degrees of freedom of movement were possible using a Dualshock 4 playstation controller – two for each of the two joysticks.

The scene also needed to be set up in such a way that CARV could generate a good reconstruction on the operator site. This involved placing several small objects typically found in an office in the scene, for example, desks, chairs, and tables, at various distances from the camera. Our reasoning was to emulate what Lovi et al. (2011) used to obtain their results, i.e. an office scene. Furthermore, since our scene was a simulation, the objects needed to be textured with patterns such that ORB-SLAM could robustly track points at different depths as the camera moved around. The scene also had to include the interactable objects. We used three cuboids placed on the table, as visible on the right side of the screenshot from the Unity scene in Figure 2.

Overall, Figure 2 shows the detailed working of the remote site in Unity. Input commands are performed by the robot immediately as they are received from the operator, resulting in a change in the mounted camera pose. The camera pose is sent immediately to the operator, but the images need to be sent in a delayed stream. Since in our setup both the remote and operator sites are on the same system, this delay needs to be simulated. To do this, instead of rendering the current view of the camera onto the Unity window, we insert it into a frame queue. At the start of the Unity scene a coroutine is used to wait for the required amount of time before dequeuing the frames in order. Once dequeued, they are sent to the operator site. For example if the required delay is four seconds, for the first four seconds no images are sent to the operator, and then onwards the stream received by the operator is the images in the same order, but those that were taken four seconds prior.

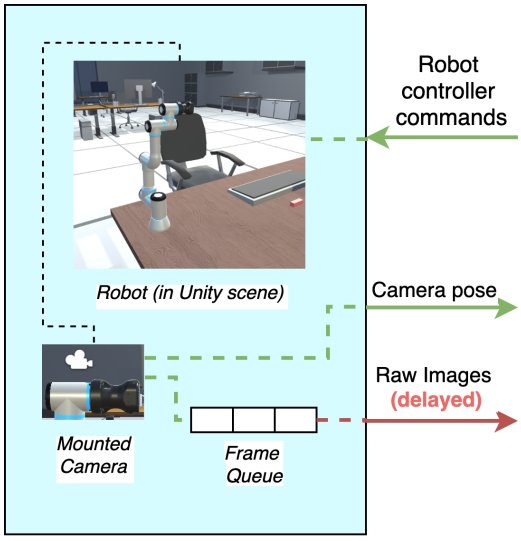

Figure 2. Detailed remote site of our teleoperation setup.

Communication – ROS

We used the Robot Operating System (ROS) to send the required information from Unity to CARV. Figure 3 depicts how ROS facilitates this communication. The circular objects are ROS package nodes, each of which needs to register with the central master server on the same IP address. We used the localhost at port 9090, since both the remote and operator sites are running on the same system. The blue nodes on the left publish the camera pose (without delay) and the camera images (with delay) to separate topics, and the corresponding subscribers on the operator site subscribe to these topics. We send the camera pose message data as a string of float values separated by underscores that is then parsed on the operator site. The images are encapsulated in headers containing the timestamp, and the format needs to be specified. The image data needs to be compressed before sending, otherwise the communication channel gets saturated and images reach the operator much slower than intended, resulting in framerate issues. We perform compression in the Unity image publisher script, but it can also be done using the ROS image transport package.

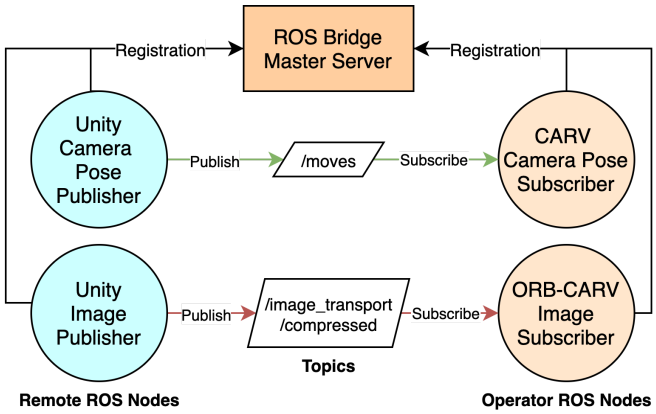

Figure 3. Diagram showing how ROS is used for communication between the two sites..

Operator Site – Predictive Display

CARV in its original configuration simply takes frames from the ROS listener and proceeds to execute 3D reconstruction. We modified CARV to instead take inputs from the topic shared with Unity, so that the Unity scene is reconstructed. We also modified it so that the camera pose used for rendering the textures in the map viewer is continuously updated with the Unity camera's translation matrix, thereby aligning the motions of the two cameras and enabling the remote-user to view the 3D structure of the scene. Figure 4 elucidates the remote-user's setup, and the controls offered to them.

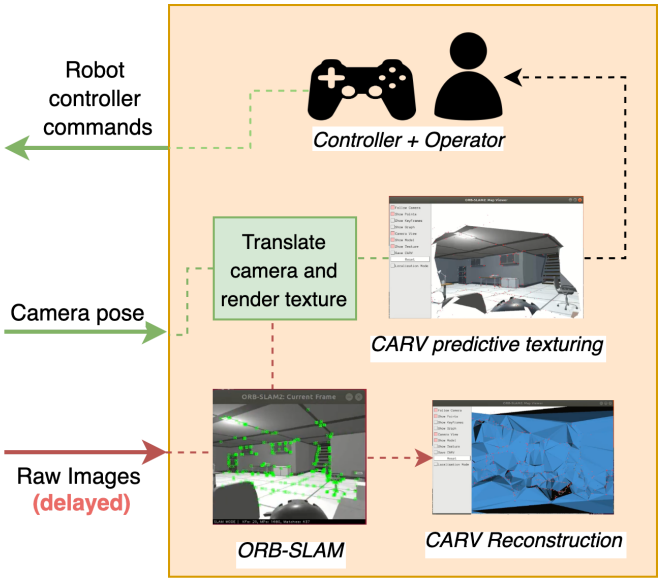

Figure 4. Detailed operator site of our teleoperation setup.

The operator site receives the Unity camera pose in real time, which moves the camera in CARV to the same degree and achieves predictive display. Based on this predicted view rendering, the user can give the next controller command to move the robot in Unity.

In order to support the aforementioned modifications, the following changes were made:

1. A ROS listener was set up to receive the Unity camera's translation matrix, and sent to CARV. The ROS listener was set up in the file "ros_mono.cc", which contains the code that sets up the entire ORB-SLAM and space carving systems.
2. The "*modeldrawer.h"* and "*modeldrawer.cc*" were modified so that at the time of texture rendering, the translation matrix of the CARV/ORB-SLAM camera is set to value received from unity.
3. Then, the textures will be rendered on to the triangulated surfaces from the updated live position of the camera. Therefore, the geometry of the predictive rendering should match that of the remote site.

Evaluation

In this section we describe how we evaluated our implementation. Again, there are three subsections, one for each component.

Unity robotic arm and scene

The evaluation for the Unity robotic arm implementation is straightforward. We just needed to ensure that the arm can reliably respond to the input commands and interact

with objects. Also, we needed to appropriately set the scene up so that a good reconstruction could be made. We needed to add multiple small objects at various depths, and also needed to make sure that the objects, especially the walls and floors, had distinct yet realistic textures so that the ORB-SLAM trackers would be accurate. This involved some trial-and-error, since the default Unity objects are too plain and we needed to apply more distinct patterns on to the objects. Another thing we needed to ensure was that the delay was being simulated correctly, by manually confirming the desired delay duration is the same as the time it takes for the delayed view images to be rendered in the Unity editor itself.

ROS Communication

We also had to ensure that the ROS communication works as expected. This was done by confirming that images were being sent from Unity and received by CARV as the camera moved around the scene. Also, we confirmed that the delay duration in receiving the images on the operator site was the same as that specified in Unity.

Predictive Display

In “Predictive Display” mode in Unity, we simulate a delay between the time an input is given by the user, and the time that the actual view after the robot arm movement is shown. That is, there is a delay between the actual live feed and what the remote-user sees. The duration of the delay was varied to observe what effect the duration has on performance. The task for remote operation involved picking up 3 red cuboid blocks from the table the camera and robot arm are placed on, and placing them on the cabinet near the table. We opted for this task to demonstrate that delays in even such a simple task can hamper operators, and to highlight the advantage of using predictive display. The following delays were studied: 0.5s, 1s, 2s, and 4s. Furthermore, we implemented another mode of delayed video transmission, called “Reconstruction Done” mode, wherein after a reasonable 3D model of the scene is obtained, Unity sends a new texture in spurts of 0.5s, 1s, 2s, and 4s per experiment. Thus, the CARV model must rely on the saved model and graphics textures to render the view of the scene the camera faces.

The following sets of experiments were conducted and the observed times were averaged over multiple trials by both group members.

1. The time taken to accomplish the task without any delay in Unity, i.e. the user gets the live feed without any delay.
2. The time taken to complete the task with delay in Unity, i.e. the user performs the task using video feed from the camera delayed by 0.5,1,2, and 4 seconds.

3. The time taken to complete the task with delay in unmodified CARV, i.e. the user conducts the task using the CARV reconstruction of the remote site from the camera delayed (and sent to CARV) by 0.5,1,2, and 4 seconds.
4. The time taken to complete the task with the predictive view in our modified version of CARV, wherein the real-time camera translation matrix is sent from Unity to CARV, along with the video feed delayed by 0.5, 1, 2, and 4 seconds.
5. The time taken to complete the task in “Reconstruction Done” mode with our modified version of CARV, wherein a new texture is sent to the user in spurts of 0.5 and 1 seconds.

Quantitative Results

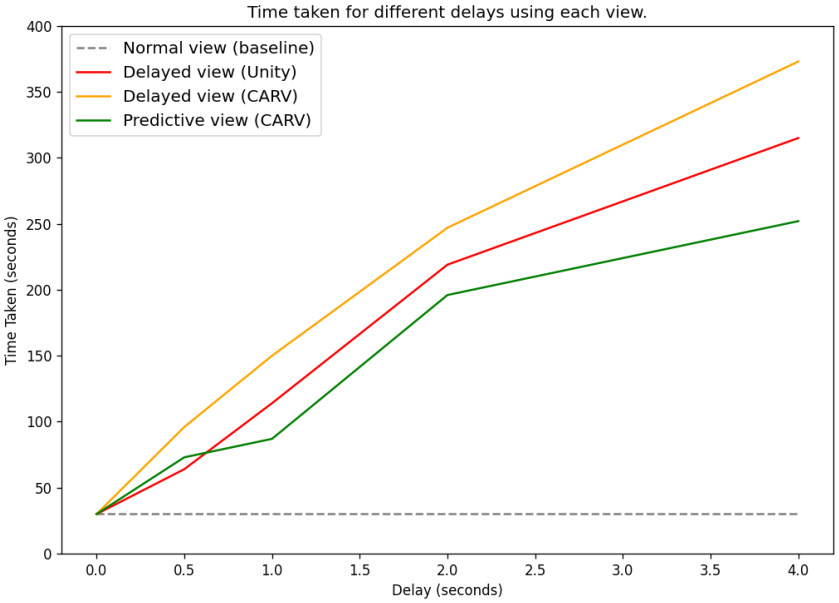

Fig 5. Plot of our experimental results showing time taken for the task using each view for different delay durations.

Figure 5 summarizes the results we obtained. The x-axis represents the delay under which the experiment was conducted, whereas the y-axis represents the time it took to conduct each experiment. The gray line is the normal view baseline, representing that it takes about 30 seconds to complete this task if the operator was actually at the remote site. The red line represents the time taken based on the delayed view. Higher delay durations lead to more time taken, as expected. The yellow line represents experiments using the CARV reconstruction, but without predictive display. A similar trend with respect to time taken as the delayed view is found. However, the performance is worse with this view, and the operator is better off using the delayed images themselves rather than CARV’s delayed approximation of what the scene looks like. The green line is

obtained from the experiments using predictive display, and it shows that using predictive display can significantly reduce the time taken for this task. Beyond 1 second of delay is where the performance improvement becomes more apparent. At 4 seconds of delay, the most benefit is observed because tasks get very difficult to do with that much latency. We believe that experiments involving more complex tasks will produce the same observations but with even more pronounced effects. This would further highlight the benefit of predictive display.

Unfortunately, the “Reconstruction Done” mode of first getting a reconstruction from a continuous delayed sequence of images and then spacing out the later images works poorly for even small delays (not plotted). This is because the underlying ORB-SLAM algorithm is unable to reconcile frames that have too much motion in between them, so trackers are readily lost and the map updates are often erroneous. As a result, the predictive display visualization also suffers. Had this been effective, our implementation would have also saved bandwidth (by sending e.g. 1 frame every second rather than 30) in addition to alleviating delay. Therefore we recommend only using the regular predictive display mode.

Conclusion

In this project we implemented a predictive display algorithm and simulated a teleoperation setup. We used Unity for the simulation and CARV for the 3D reconstruction and predictive texturing, with ROS facilitating communication between the two sites. We conducted experiments on a simple task under various delay durations. Our results show that a simple predictive display implementation can help improve teleoperation performance by reducing the time taken to complete the task, especially when the delay duration is substantial. For more complex teleoperation tasks, the same delay durations are expected to deteriorate performance even more, so our work would likely be even more advantageous. Our simulated system can run efficiently on one system without the need for a GPU. It isn’t computationally expensive because of the efficiency of CARV’s incremental free-space carving algorithms, the efficient way the predictive texturing is done without incurring additional computation costs, and the efficient communication provided by ROS. Our implementation is modular, with the predictive display implementation mainly affecting just the model drawer component, and our code repository is detailed enough so that it should be straightforward for further research to easily build upon our work.

Future work

Although the results are good and our predictive display can help alleviate the delay in teleoperation, our work can likely be improved upon. We are using a simple translational

warp for the predicted view because it works well for this task. Our system can be used as an easy way to test out different types of texture warps for different implementations of predictive display as well, which hopefully will aid future research in this direction. For example, if the task requires more degrees of freedom of movement of the robot, the warp can incorporate rotation information as well for better visualization. Further, improvements can be made to the underlying CARV system. Semi-dense CARV, for example, may provide better visualization as it is more robust to noise.

Acknowledgements
We would like to thank Prof. Martin Jagersand and the two TA's Islam Ali and Chen Jiang for their invaluable guidance and feedback throughout this project. We also thank Prof. Nilanjan Ray for providing access to one of the systems in the Vision and Robotics Lab on which we implemented our project.

References

Cobzas, D. and Jagersand, M. (2005). Tracking and predictive display for a remote operated robot using uncalibrated video. IEEE Conference on Robotics and Automation (ICRA), pp 1859-1864. Best vision paper award.

Cheung, K. M. G., Baker, S. and Kanade, T. (2003). Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings., 2003, pp. I-I, doi: 10.1109/CVPR.2003.1211340.

He, S., Qin, X., Zhang, Z., and Jagersand, M. (2018). Incremental 3D line segment extraction from semi-dense SLAM. 1658-1663. 10.1109/ICPR.2018.8546158.

Jin, J., Petrich, L., He, S., Dehghan, M., and Jagersand, M. (2019). Long range teleoperation for fine manipulation tasks under time-delay network conditions. CoRR, abs/1903.09189. http://arxiv.org/abs/1903.09189

Klein, G. and Murray, D. (2007). Parallel Tracking and Mapping for Small AR Workspaces. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 225-234, doi: 10.1109/ISMAR.2007.4538852.

Lovi, D., Birkbeck, N., Cobzas, D., and Jagersand, M. (2011). Incremental free-space carving for real-time 3D reconstruction. https://webdocs.cs.ualberta.ca/~dana/Papers/103dpvt_Lovi.pdf

Rachmielowski, A., Birkbeck, N., and Jagersand, M. (2010) Performance Evaluation of Monocular Predictive Display. In ICRA, pp 5309–5314.