LIVE FEEDBACK ON A LIVE VIDEO USING HTML5 CANVAS

By

Sandeep Devarapalli

Bachelor of Technology in Information Technology

Jawaharlal Nehru Technological University

Hyderabad, Telangana, India

2013

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE December, 2016

LIVE FEEDBACK ON A LIVE VIDEO USING HTML5 CANVAS

Thesis Approved:

Dr. Christopher Crick

Thesis Advisor

Dr. David Cline

Dr. Johnson Thomas

Name: Sandeep Devarapalli

Date of Degree: January, 2016

Title of Study: LIVE FEEDBACK ON A LIVE VIDEO USING HTML CANVAS

Major Field: Computer Science

Abstract: The objective of this research is to teleoperate a robot situated at a remote location with the help of video camera. The operator relies on live video to move the bot and perform different tasks, but in the case of laggy video the operator has to wait until the video is received which is frustrating and tiresome. We intend to solve this by giving feedback to the operator instantly as the command is given. HTML5 Canvas gives us flexibility as it can be used to access pixel values from the video and manipulate them giving feedback to the operator on the video itself. Preliminary data shows a 30% decrease in time for doing certain tasks using this system instead of without using the system compared to the standard video approach.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Teleoperation is defined as operation of a machine at a distance. It is similar in meaning to the phrase “remote control" but is usually encountered in research, academic and technical environments. Teleoperation is controlling a robot from a distance. Distance can vary from tens of centimeters to millions of kilometers. A teleoperation system consists of a master device that the operator holds on to and a slave device, the robotic tool at a network location. Since the user cannot see the robot directly, he or she must rely on feedback from the robot's worksite. This is presented to the user by way of the interface. Different forms of feedback in teleoperation are

- Live video from video cameras
- Haptic (touch, such as a vibration)
- Auditory (human ear range)
- Temperature
- Contact sensors
- Sonar images

Applications for a teleoperated bot could be in various fields such as space exploration, nuclear reactor maintenance, bomb diffusing or medical surgery. In the case of space exploration, activities are fundamentally limited by the amount of energy required to raise loads into earth orbit. An additional requirement, when humans go into space, is the expense and additional mass of life support and safety critical systems. Because of

these reasons, conducting operations such as protein crystal growth on manned space missions is very expensive. Teleoperation technology can have a substantial impact on the cost of microgravity operations by reducing the number of humans required in space for a given amount of work. Difficulties in this system include communication time delay, restrictions of communication capacity and limitations of computation power on board. These space systems demand a high level of safety and reliability.

The main control problem in teleoperated systems is the instability induced by the communication time-delay and incomplete information on both sides (master and slave). Existing systems rely on video which is transferred over the network and send to the master to teleoprate. During situations of lag the master has no clue if his next command has executed or not; he has wait until the video is received. We propose using HTML5 Canvas [1][3][6][8] to show the video which allows for interesting video manipulations. In this work we modify the video feed, giving instant feedback to the master on the video itself so that the master knows all the commands which are going to be executed and when to give new commands. HTML5 Canvas can be used in drawing interactive video and audio, dynamic graphics and animations.

HTML5 Canvas Video [1][3][6][8][9][16]: Even though the video is displayed only on HTML5 Canvas, you still need a <video> tag in HTML. The key is to put the video in a (or a similar construct) and to set the display CSS style property of that to none in HTML. This will ensure that while the video is loaded in the page, it is not displayed.

Example code

```
<script>
    var v = document.getElementById('v');
    var canvas = document.getElementById('c');
```

```
    var context = canvas.getContext('2d');
    v.addEventListener('play', function(){
        draw(this,context,cw,ch);},false);},false);
    function draw(v,c,w,h) {
         if(v.paused || v.ended) return false;
         c.drawImage(v,0,0,w,h);
         setTimeout(draw,20,v,c,w,h);
}</script>
```

When we write code to display a video on the canvas [10], you use the c.drawImage() function, as though you were displaying a static image. We need to call drawImage() in some sort of loop to continually update the image with new data from the playing video in the HTML page (hidden or not). To do this, we call the video's play() method and then use a setTimeout() loop to call the draw() function every 20 milliseconds.

The Robot Operating System (ROS) [12][13][14] is a collection of software framework for robot software development, (see also Robotics middleware) providing operating system-like functionality on a heterogeneous computer cluster. ROS [14] provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages.

Despite the importance of reactivity and low latency in robot control, ROS itself is not a Realtime OS, though it is possible to integrate ROS with realtime code.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server.

Rosbridge [12][14] is a middleware abstraction layer which provides robotics technology with a standard, minimalist applications development framework accessible to applications programmers who are not themselves roboticists. Rosbridge provides simple, socket-based programmatic access to robot interfaces and algorithms provided by ROS. In particular, rosbridge facilitates the use of web technologies such as Javascript for the purpose of broadening the use and usefulness of robotic technology. Rosbridge provides an additional level of abstraction on top of ROS.

Rosbridge treats all of ROS as a “back end”. This shields application developers from needing intimate knowledge of low-level control interfaces, middleware build systems and sophisticated robotic sensing and control algorithms. At a bare minimum they must understand the build and transportation mechanisms of the middleware package, however rosbridge layers a simple socket serialization protocol over all of this complexity, on top of which application developers of all levels of experience can create applications. ROS abstracts individual robot capabilities, allowing robots to be controlled through messages. It also provides facilities for starting and stopping the individual ROS nodes providing these capabilities. Rosbridge encapsulates these two aspects of ROS, presenting to the user a unified view of a robot and its environment.

The Rosbridge protocol allows access to underlying ROS messages and services as serialized JSON objects, and in addition provides control over ROS node execution and environment parameters.

The TurtleBot is a low-cost, personal robot kit with open-source software as shown the Fig 1.1. With TurtleBot, you can to build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications. One of the most striking features is the camera, which allows you to retrieve the data coming from the Kinect sensor which includes an RGB image, depth image, and point cloud data as well. We are using the TurtleBot to get a live video stream which we are using as an implementing platform for our thesis.

TurtleBot

A Mobile Base and Power Board

- iRobot Create
- 3000 mAh Ni-MH Battery Pack
- 150 degrees/second Single Axis Gyro
- 12V 1.5Amp Software Enabled Power Supply (for powering the Kinect)

B 3D Sensor

- Microsoft Kinect
- Kinect Power Board Adapter Cable

C Computing :: ASUS 1215N

- Processors :: Intel® Atom™ D525 Dual Core Processor
- Memory :: 2 GB
- Graphics :: NVIDIA® ION™ Discrete Graphics Processor
- Internal Hard Drive :: 250 GB

D TurtleBot Hardware

- Kinect Mounting Hardware
- TurtleBot Structure
- TurtleBot Module Plate with 1 inch Spacing Hole Pattern

Fig 1.1 TurtleBot

CHAPTER 2

REVIEW OF LITERATURE

During the last decade, various teleoperation robotic systems had been developed to allow human operators to execute tasks remotely in a variety of applications such as tele-manufacturing, tele-healthcare, tele-mining and tele-surgery. These systems include sophisticated robotic technology, equipment and interfaces to perform tasks remotely that are normally too dangerous for humans or are within hostile environments. The equipment used and the cost to build such systems is quite high and many still view these system as not being practical or reliable.

With the rapid advancement in Internet and robotic technology, Internet-based or Web-based teleoperation is gaining popularity in the robotics field. Despite all the achievements on Internet-based teleoperation robotics, a number of research and technical challenges still need to be addressed before teleoperation applications can be achieved in real environments. Some of these challenges include (but not limited to) issues related to actuators and sensors, restricted Internet bandwidth, arbitrarily large transmission delay, lost packets, disconnection and issues with user interfaces.

With the advent of ROS (Robotic Operating System) [12][13][14], an open source robot middleware, together with other supporting components, such as rosbridge [13][14], mjpeg server [12][13][14]and rosjs [13][14], it has now become relatively simple to implement Internet-based teleoperation. One advantage is that ROS can utilise state-of-

the-art navigation and perform manipulation algorithms on standard robot platforms. In brief, ROS administrates the low-level robot controllers, sensor processing, planning, navigation and manipulation. ROS provides libraries and tools for implementing robot applications. Rosbridge manages communication channels between Web applications and ROS. Rosjs handles rosbridge connection on the Web application side. It uses WebSockets [14] to connect with the rosbridge and provides publishing, subscribing, service call and other essential ROS functionality. Finally, the mjpeg server can be used to stream video efficiently to the Web.

ROS, Rosbridge and rosjs make it much easier to port configurations from one robot to another, to design and instantly test robot interfaces, to leverage algorithms developed elsewhere, to engage large numbers of experimental subjects over standard web interfaces, and to apply arbitrary programming languages, tools and libraries to problems in robotics and HRI (Human Robot Interaction) [13].

In [6], Laurent Denoue, Scott Carter, Matthew Cooper describe new direct video manipulation techniques enabling users to quickly copy and paste content from video documents into a user's own multimedia document. While the video plays, users interact with the video canvas to select text regions, scrollable regions [11], slide sequences built up across many frames, or semantically meaningful regions such as dialog boxes. Instead of relying on the timeline to accurately select sub-parts of the video document, users navigate using familiar selection techniques such as mouse-wheel to scroll back and forward over a video shot in which the content scrolls, double-clicks over rectangular regions to select them, or clicks and drags over textual regions of the video canvas [10] to select them. Laurent Denoue, Scott Carter, Matthew Cooper describe the video processing techniques that run in real-time in modern web browsers using HTML5 and JavaScript; and show how they help users quickly copy and paste

video fragments into new documents, allowing them to efficiently reuse video documents for authoring or note-taking.

Video is embedded as an HTML5 video and JavaScript paints incoming frames in a CANVAS element [4][5][6][8]. Pixel data [9] is manipulated in JavaScript to perform real-time video and document processing such as binarization, frame differencing, line-detection, connected component analysis and scroll detection across frames. A timer is used to continuously draw incoming frames onto a CANVAS element that the user sees. JavaScript event listeners are attached to this element and trigger the different actions: double-click triggers region detection, mouse-wheel triggers scroll-analysis, and click and drag triggers text detection.

At this point, if the user initiates a Copy command using the keyboard or other means, the system automatically generates a PNG image out of the cropped area from the video canvas and copies it to the clipboard. If instead the user starts a left or right drag operation over the canvas [11], the system draws previous or next frames, depending on the drag direction. When the rectangular region is no longer detected at the same location, the system stops showing previous or next frames. At this point, if the user issues a Copy action, the system generates an animated GIF of all the frames found between the start and end time in the video sequence where that rectangular region was shown.

In [7], Manh Duong Phung, Thuan Hoang Tran, Thanh Van Thi Nguyen and Quang Vinh Tran propose using multiple transport protocols for the control of internet based robots. A TCP connection is opened when a teleoperation session is started and is closed once the teleoperation is geared up. RTP protocol is employed for the transmission of information on live scene images. UDP is used for sensory data and robot positions and speeds.

Since the human operator issues control commands based on his/her personal judgments and decisions, the timing of these control signals is random in nature and is controlled solely by the human operator. The transmission frequency of control signals depends on how often the user interacts with the mobile robot. As a result, Manh Duong Phung, Thuan Hoang Tran, Thanh Van Thi Nguyen and Quang Vinh Tran still utilize UDP for control command delivery. The usage of UDP is generally acceptable because control commands are small-size packets and it should not jeopardize inter-flow fairness if the human operator does not issue a burst of commands when the network is heavily congested.

In [2], Dilip Kumar Limbu, Wong Hong Vee Alvin, Chua Yuanwei, Albertus Hendrawan Adiwahono, Tran Anh Dung, and Han Boon Siew implemented Internet-based teleoperation similar to our paper but their output is different. The objectives of the study were to: a) test the usability of TeleBot (TurtleBot with JACO arm), b) compare two commonly used devices to control TeleBot, and c) understand further development and requirement of TeleBot by examining how operators use TeleBot to complete a specific task. A task-oriented user study that included navigating TeleBot to a specific location and using TeleBot's arm to touch an "X" marking placed on the door, was designed. Eight participants performed the same task using two different types of device. Both objective and subjective data were collected during the study. The main results of the study here a) TeleBot demonstrated a certain level of usability; b) the gamepad was slightly easier to use for teleoperation when compared to keyboard and finally c) to deduce further developmental area for TeleBot.

In [15], Andrew Wichmann, Burcu Demirelli Okkalioglu, Turgay Korkmaz investigate Wireless Sensor Networks (WSNs), Mobile Robotics, and Teleoperation. Due to the complementary nature of these areas, there is a growing trend in integrating them to support various applications that can go beyond passive monitoring and allow us to actually interact with the environment through autonomous and teleoperated robots. Such applications can extensively be used in medicine, science, military, industry, nuclear power stations, underwater, and space explorations. In this paper, Andrew Wichmann, Burcu Demirelli Okkalioglu, Turgay Korkmaz start with a review of the history of WSNs, robotics and mobility in WSNs, and teleoperation. They then introduce a system model integrating these areas as Wireless Sensor and Robot Networks with Teleoperation capabilities (WSRNT). Using this model, they define the problems associated with WSRNT [3].

One of the key challenges in WSRNT is how to manage the decision making among the components of such an integrated system. The decision making process can be divided into three categories: autonomous, teleoperation, and hybrid. Autonomous decision making is a distributed approach that leaves the decisions up to the networks to decide which robots will respond to which requests by the sensors. Teleoperation decision making is a more centralized approach because it uses a human operator to control the actions of the robot(s). Teleoperation allows operators to actually interact with the environment, as opposed to passively monitoring it. A hybrid approach tries to combine autonomous decision making with teleoperation in an efficient way. More specifically, an operator can use one or more robots to accomplish a task through teleoperation while the other robots autonomously conduct data collection or perform event monitoring.

In [3], Andrew Wichmann, Turgay Korkmaz analyze different deployment and movement policies in wireless sensor and robot network. Mobile robots [15] have been used to collect data in the network in order to reduce the energy consumed when forwarding data. In order to be energy efficient, mobile robots should be deployed and controlled in such a way as to minimize travel distance. To do this, Andrew Wichmann, Turgay Korkmaz examined multiple deployment and movement policies and their effects on the performance of a network. Andrew Wichmann, Turgay Korkmaz tested three different deployment schemes and analyzed the results. Andrew Wichmann, Turgay Korkmaz also analyzed the expected distance to each event to discover what, if any, effects the initial deployment has on that metric and in turn compared this to previous results. Andrew Wichmann, Turgay Korkmaz then tested three different collection methods and analyzed their performance. Next, they added service times to their model and lastly they changed the distribution of events to examine how these changes affected the network performance.

Existing teleoperating system would rely upon the live video to be reached to the operator and as the network is laggy it would a lot of time for the video to reach. This is frustrating and tiresome for the operator. We address this problem by giving instant feedback to the user on the video itself instead of waiting for the video. The input is measured during the lag time and immediate feedback is given to the user acknowledging that his/her input has been received. As the existing system relied upon normal video players to show the video which can’t be used to manipulate video frame. Our system shows the video in HTML5 Canvas which helps in getting access to each individual frame and which can be manipulated to suite our need.

CHAPTER 3

METHODOLOGY

With the help of literature review few important point must to be taken into consideration before writing an algorithm. First, real-time response between pressing the keys/controls and robot reaction is very important. Second, placement of cameras in the environment that is being captured with reference to the robot's overall structure and positioning is equally important. Third, near real-time video streaming is needed to be shown on the screen monitor. Latency is the amount of time it takes for a single packet to traverse a network. It is normally expressed as a round-trip-time (RTT) which is the amount of time it takes a packet to go from A to Z and back again. Lag is the Quality of Experience (QoE) impact on different network issues, resulting from latency. It is the delay experienced by a user when they press a button or drag a window and have to wait for something to happen.

In our case due to the bad network the live video is laggy which may be accompanied by high latency. In order to have best experience in teleoperating the bot, the operator need to know when the video is laggy and when he can give commands. We intend to do that by giving constant feedback on the video itself, when the operator gives command and when the video frame actually changes. For example if the operator wants to pan left he presses the left arrow key, simultaneously the video pixel [9] on the canvas [10] are manipulated to give the operator feedback of his command, but due to the lag

in the network the operator won't be able to see the change there itself. While teleoperating a complex task this feedback system works constantly helping the operator in achieving the task.

Algorithm 3.1: Panning Left or Right

Input: Keyboard input by the user (C).

Output: Manipulated frame shown on the monitor when giving the input.

Procedure:

```
1:  d1=ctx.getImageData(0,0,width,lenght);        *Getting actual frame
      pixel values
2:  for i=0 to d1.lenght do                       *For loop continues till the end of all
                                                    pixels
3:    for j=i+queue to i+width do                 *For loop for getting pixels in
                                                    horizontal line.
4:    r=d2[j - queue]                             *Getting red pixel value
5:    d[j] = r;                                   *Manipulated frame red pixel value
6:    end for
7:    for j=i to i+queue                          *For loop for showing black pixels
8:      d[j] = 0;                                 *Showing black pixels in the
                                                    manipulated frame
9:    end for
10: end for
```

Depending upon the input given to the robot in the measured lag time the input is represented in queue, image frame is taken which contains all the pixel values. The two for loops in Algorithm 3.1 are used to traverse through all the pixel values horizontally from top to bottom and left to right. Queue is calculated depending on the input which tells how much the frame has to be turned and depending on the value the remaining pixels [9] are moved to the left side or the right side showing the manipulated image. Black pixel are used to show the traverse and when a new frame comes the original video is shown with the correct angle in which the robot turned.

Algorithm 3.2: Moving Front or Back

Input: Keyboard input by the user (C).

Output: Manipulated frame shown on the monitor when giving the input.

Procedure:

```
1:  d1=ctx.getImageData(0,0,width,length);        *Getting actual frame
2:  for i=0 to d1.lenght do                       *For loop continues till
3:      r=d[i];                                   *Getting red pixel value
4:      v=(0.2126*r + 0.7152*g + 0.0722*b>=d2[i])?255:0; *Comparing
                                                  greyscale pixels with
                                                  normal pixels and these
                                                  numbers are used to
                                                  convert normal pixel to
                                                  greyscale value
5:      d[i]=v;                                   *Contains white or black pixel
6:      if d[i] contains 255                      *If it contains white pixels
```

```
7:      d[i]=d2[i];                          *Assigning previous frame
8:    else
9:      d[i]=d2[i+8];                        *Assigning manipulated values
10:   end if
11: end for
```

Initially in Algorithm 3.2, the frame containing all the pixel values is taken and saved in r, g, b variables. Then these variable are used to compare the grayscale value of the pixels with the original pixels whose output would be a 255 or 0. If the output has 255 then the manipulated frame would have the same pixel value as the input frame and if the output has 0 then the manipulated frame would have manipulated pixel value. The for loop is used so that this is checked throughout the pixels values and manipulated frame is shown in the end.

3.3 Equations

Parameters in our experiment would be lag time, input frequency, angle of motion, number of inputs. Lag time would be measured lag of the network which is network dependent. Input frequency would be the input frequency of the keyboard from which the input is given and for our system it is 20 hertz. Angle of motion would be the angle in which the Turtlebot is moved which is a fixed value and for our system it is one radian for one second input. As the input frequency is 20 hertz which means number of input in one second would be 20, which means that the system takes 20 inputs in one second. So for each input it takes 1/20 sec as the input frequency is 20 hertz.

For each input it takes 1/20 sec which also means that the bot moves 1/20 rad for that one input. Suppose there are 2 left input then the bot would move (1/20) + (1/20) =1/10 rad. Similarly for n inputs it would be summation for n. Angle of view is the angle in which the bot would move for a given input which is represented in equation (1).

Lag time (l)-sec

Input frequency (C) =20 hertz (Keyboard refresh rate)

Angle of motion (θ) = 1 rad/sec (+1 left turn, -1 right turn)

Number of inputs (n) = 20 (for 1sec lag)

Angle of view (V) =rad (How much angle the bot show move)

For each input it takes 1/20sec to cover as the input frequency is 20 hertz

For only one left input

Angle of view (V) = $\theta/20$ rad

For all left inputs over a period of one sec where $n = 20$

Angle of view (V) = $\sum_{i=1}^{n} \theta\ /20$ ——————————— (1)

$$=\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)+\left(\frac{\theta}{20}\right)$$

$$=20\theta/20$$

$$=\theta$$

=1 rad

CHAPTER 4

FINDINGS

4.1 Experiment Setup

We set up two test environments for our system which we call course-A and course-B. For the task-oriented evaluation, four main task scenarios were designed: a) navigate the TurtleBot on course-A with and without using our system by the natural time delay of the network, b) navigate the TurtleBot on course-A with and without using our system by a delay imposed by a proxy server far away, c) navigate the TurtleBot on course-B with and without using our system by the natural time delay of the network, and d) navigate the TurtleBot on course-B with and without using our system by a delay imposed by a proxy server far away.

The experiment was conducted in an office environment (Fig 4.1). The setuop allows remote experiments to be conducted as well. The observer had little or no influence during the whole course of the experiment, and would observe most of the time and intervened only when necessary (E.g. TurtleBot was about to hit the wall). The experiment included a laptop and a TurtleBot.

Participant used the keyboard to tele-operate the TurtleBot and no other assistance was given by the observer. This allowed observation of the participant's interaction to be unobtrusive, and also maintained consistency across all of the subjects. As such, intervention during the experiment was limited only to occasions where technical

problems prevented the subject from continuing their given task. An error metric was also measured to know how often the test subjects messed up. The participants were given five minutes to learn how to use the TurtleBot and explained how to navigate course-A and course-B. After the five minutes the participants were randomly given one of the four tasks and the total time taken by each participant was measured. The random nature of assigning tasks helped in getting good results as the participants would not be able to be habituated with the course.

Course-A is going from point A to point B as fast as possible in a straight line. So the participant starts from point A and goes to point B and then come backs to point A. Course-B is different from course-A as you only go from point A to point B and would not come back. The course is also tricky as the participant has to pan a lot of times and has to make sure that he or she would not cross the wide line which act as a border. The participant has to go from point A to point B as fast as possible.

4.2 Navigating Course-A

- In this model we teleoperate a bot at a remote location from point A to point B.
- Initially the bot is facing point A which the operator see for the first time.
- Then the bot is turned in the opposite direction to face the direction of point B and reaches point B which is at a distance from point A.
- After reaching point B the bot follows the previous step again to reach point A which was the starting position.
- The experiment is done twice, once using the system, and once without using the system.
- The operator has to navigate from point-A to point-B as fast as possible.
- The best way to do this is navigating in a straight line.

<-Step 1

Step 2->

Figure 4.1: Navigating course-A

<-Step 3

Step 4->

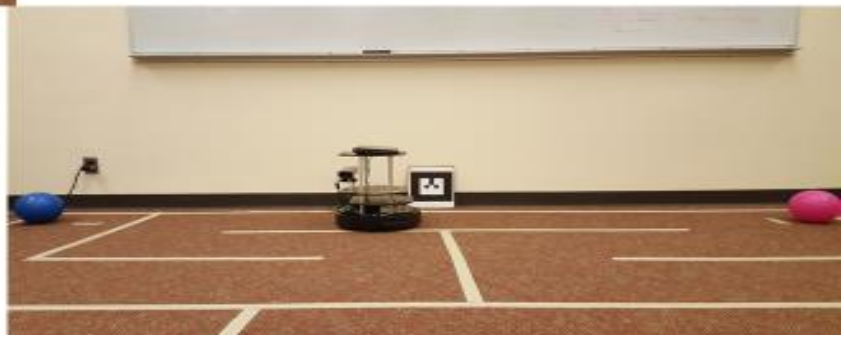
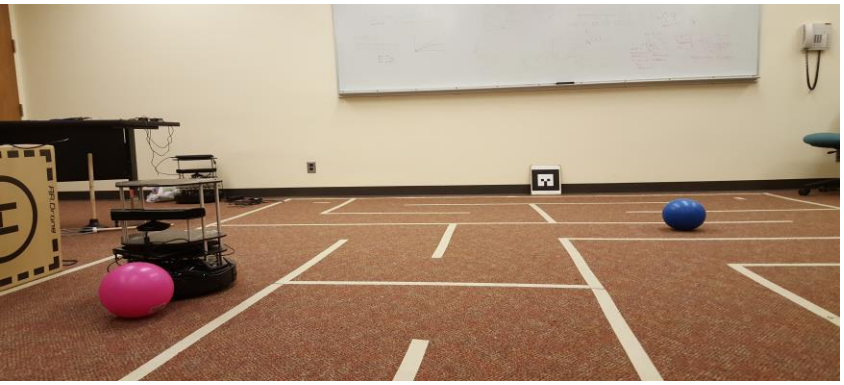


Figure 4.2: Navigating course-A

4.3 Navigating Course-B

- In this model we will teleoperate a bot at a remote location from point A to point B.
- The bot has to be navigated so that it doesn't touch the white line on the ground.
- The operator has to navigate from point A to point B as fast as possible.
- If the operator accidently touches the white line they have start the experiment again.
- The experiment is done twice, once using the system, and once without using the system.
- The operator has to navigate in from point-A to point-B as fast as possible and has to make sure that the bot won't cross the white line.



<-STEP 1

STEP 2->

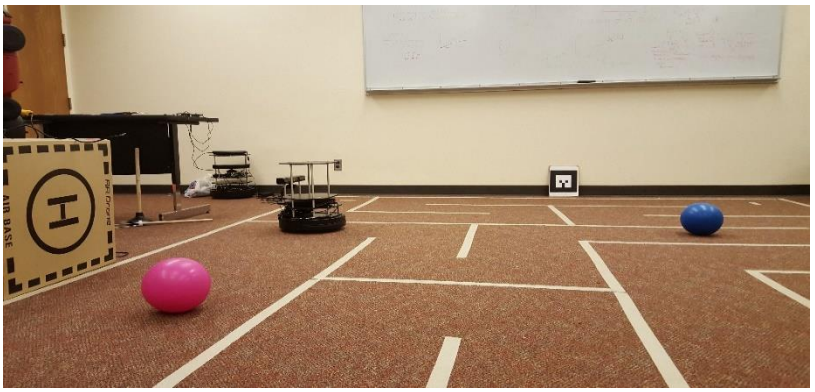
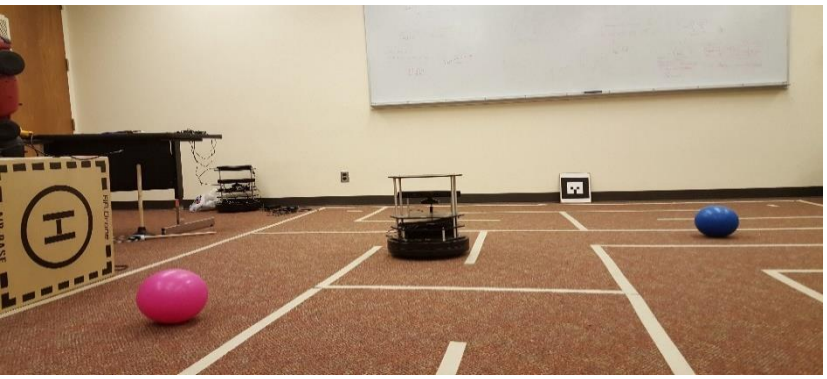
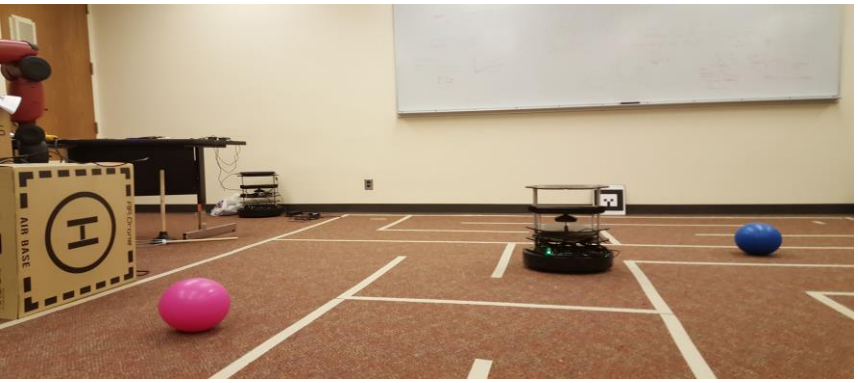


Figure 4.3: Navigating course-B



<- STEP 3

STEP 4->



<-STEP 5

STEP 6->

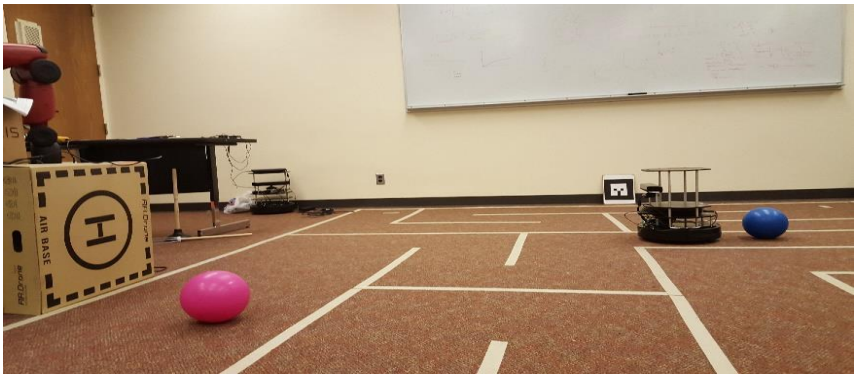


Figure 4.4: Navigating course-B

Name	Age	Location	Experience(robot/interface)
Prateek	25	Lab	Nill
Nikesh	23	Remote	Nill
Harish	23	Lab	Medium
Pradeep	24	Remote	Nill
Divya	23	Remote	Nill
Sai	24	Remote	Nill
Charan	24	Remote	Nill
Nikhil	25	Remote	Nill
Arjun	23	Remote	Nill
Vikas	23	Remote	Nill

Participant Information

4.4 Results

- Total time taken by the operator is measured against the lag time.
- The operator is give course-A or course-B randomly.
- The operator does the experiment using immediate feedback and without using immediate feedback.
- Immediate feedback would help the operator in situations where is video is laggy and reduces the overall time to do the job.
- Graphs are plotted showing total time taken and the measured lag time.
- Mean time for doing the experiment is also plotted in a particular lag time with standard error.

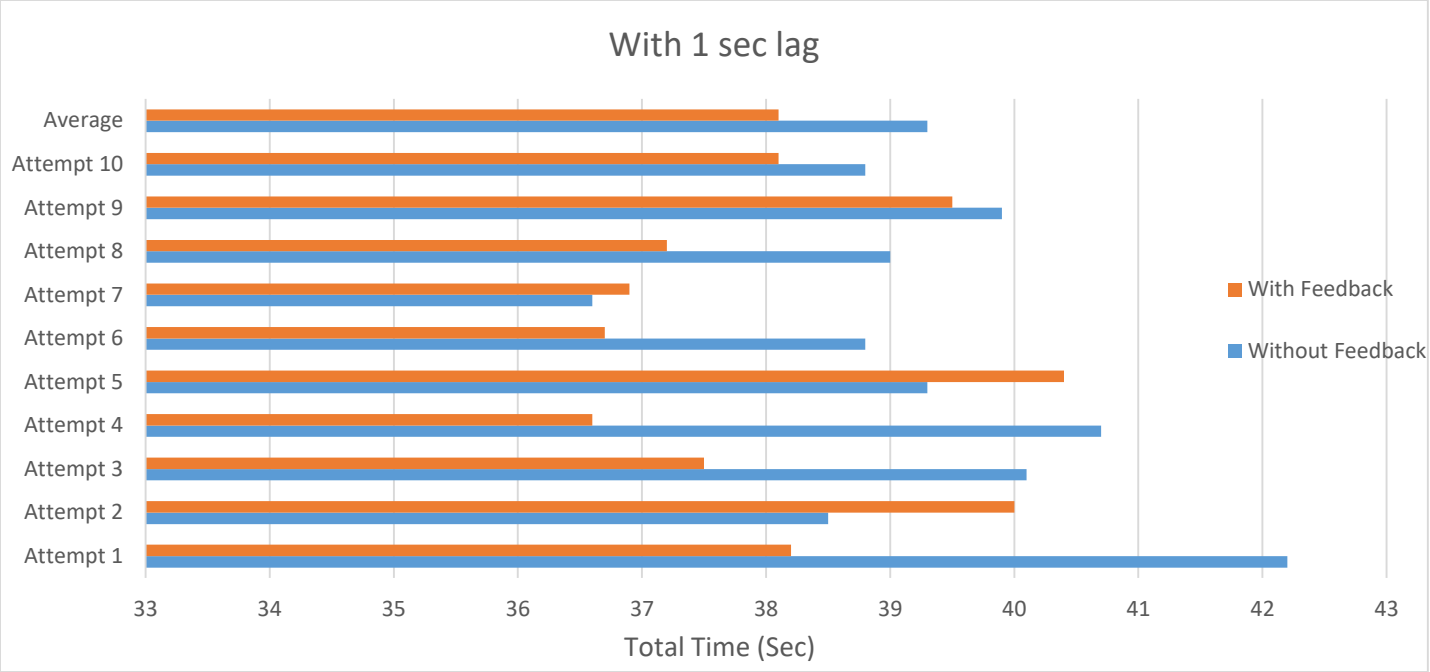


Figure 4.5: Total time taken at 1 sec lag —course-A without proxy server

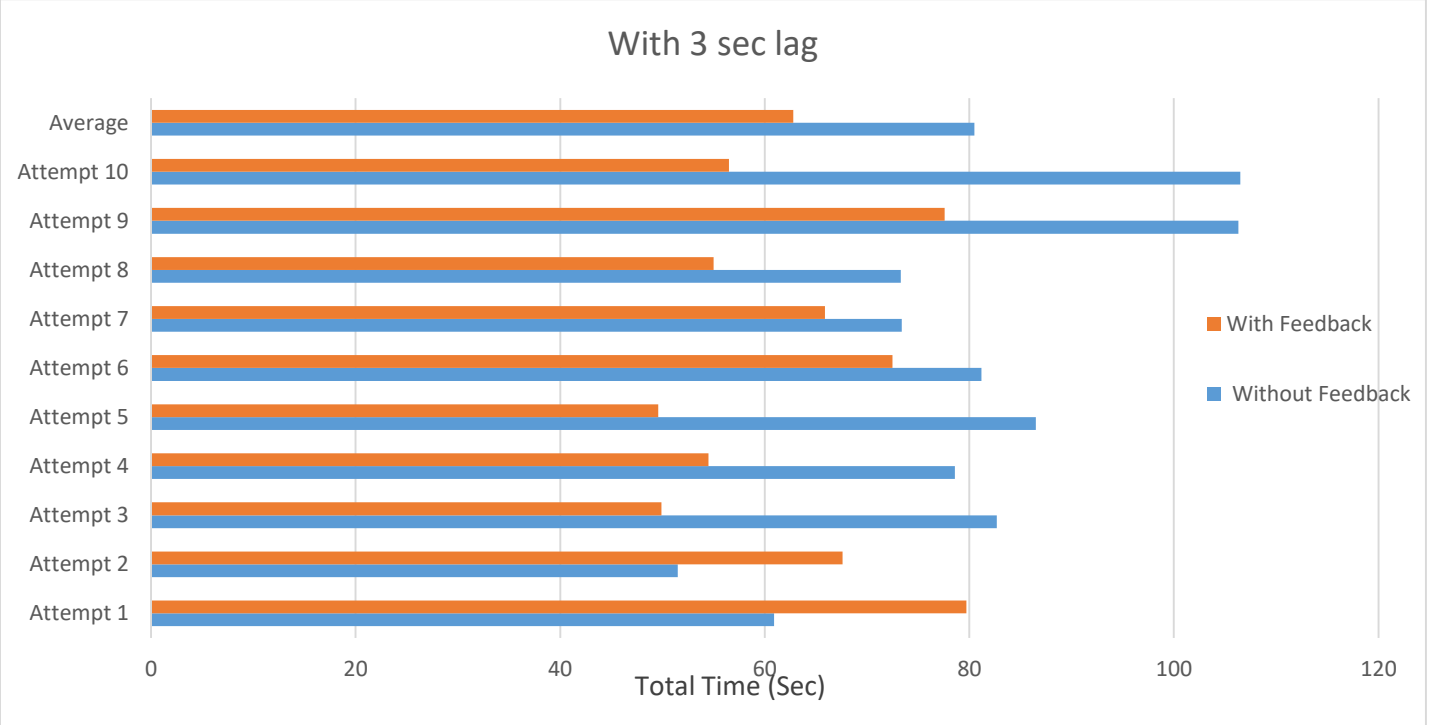


Figure 4.6: Total time taken at 3 sec lag —course-A with proxy server

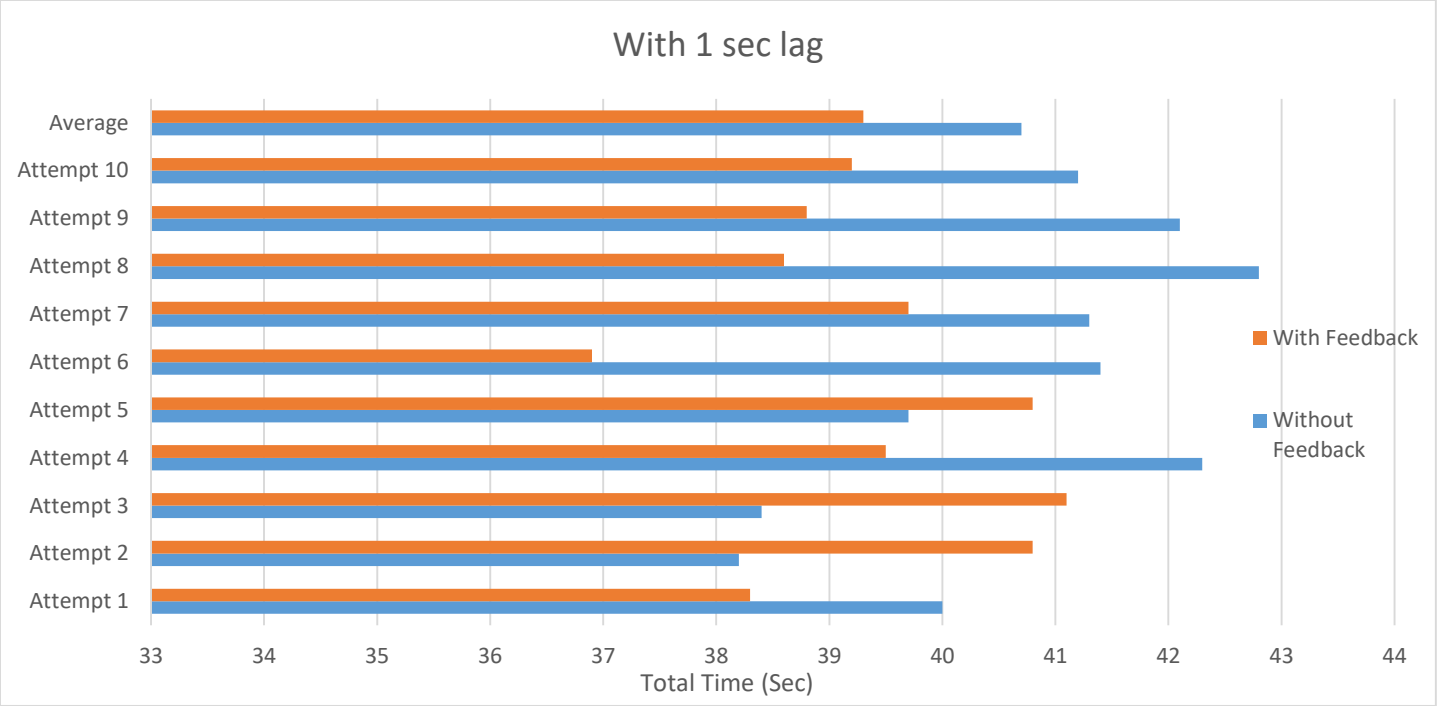


Figure 4.7: Total time taken at 1 sec lag —course-B without proxy server

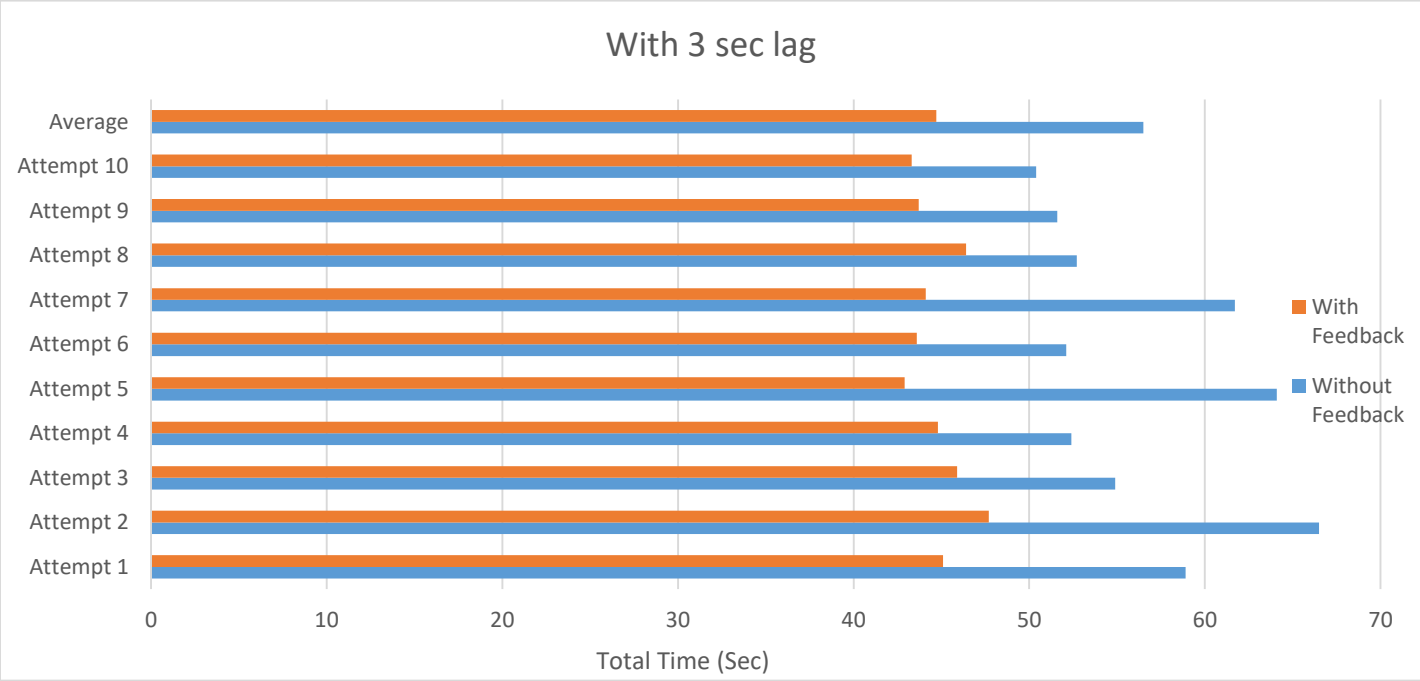


Figure 4.8: Total time taken at 3 sec lag —course-B with proxy server

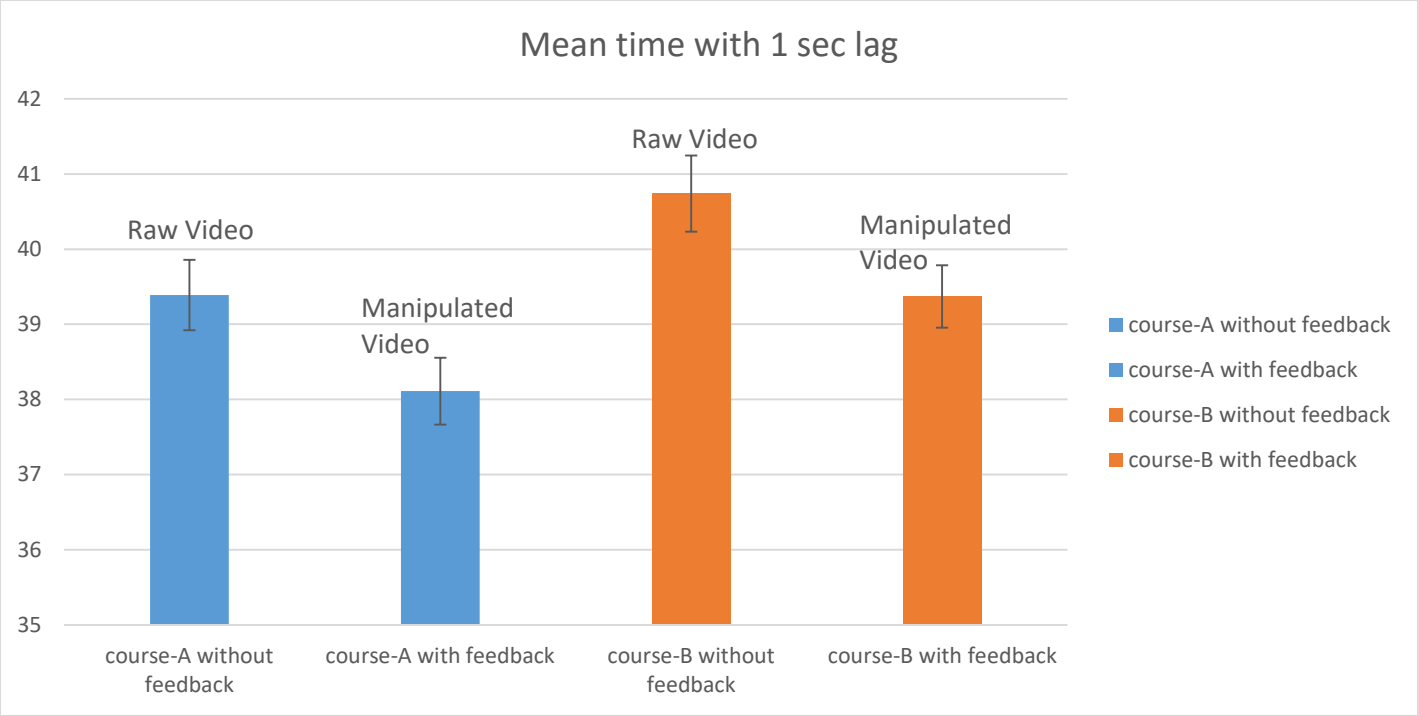


Figure 4.9: Course-A and course-B without proxy server and Standard Error

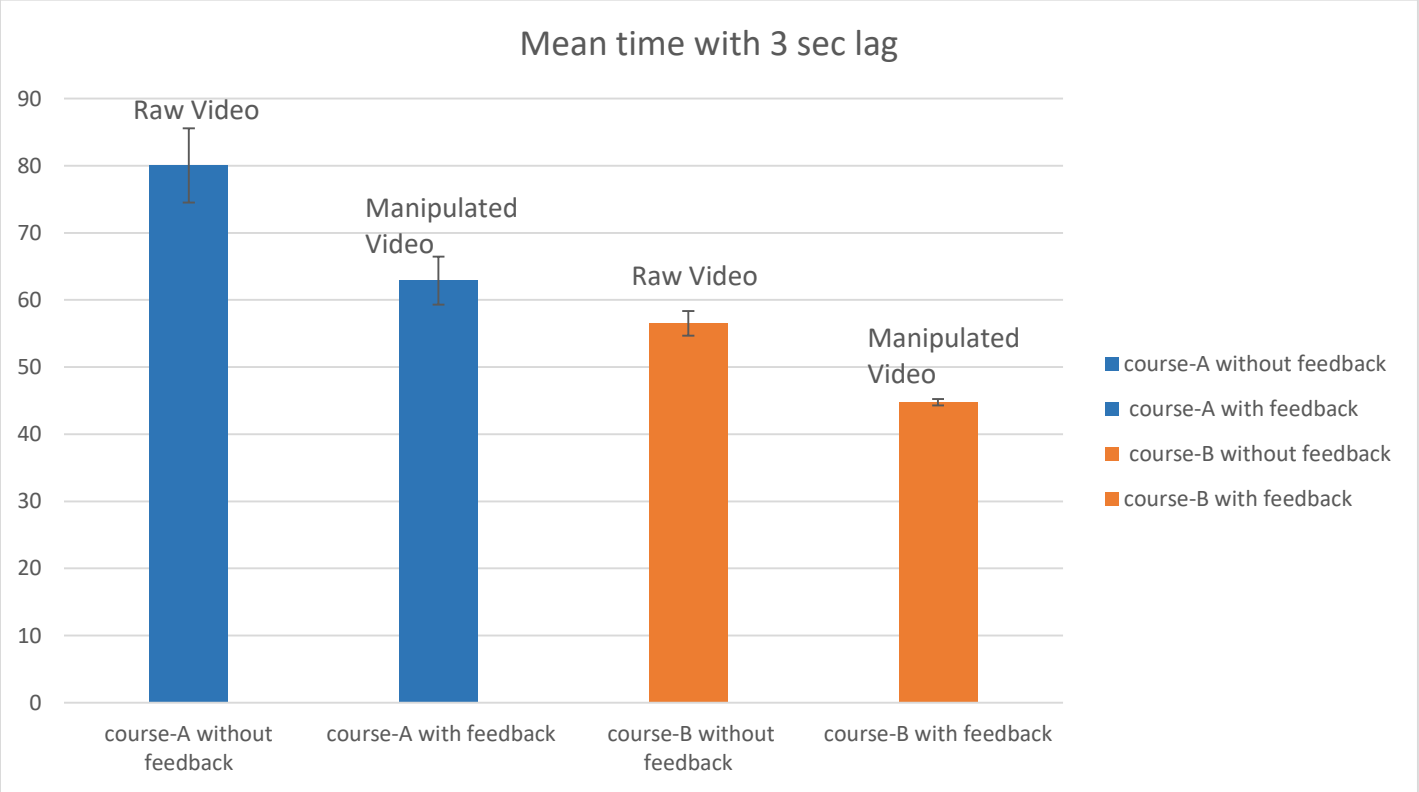


Figure 4.10: Course-A and course-B with proxy server and Standard Error

From the Fig 4.5 we can see that even though the lag is 1 sec our system works better in using feedback that without using feedback. The average time taken by using the system is less than when the system is not used. Our hypothesis that our system works well if there is more lag is true which is shown in the Fig 4.6 where a proxy system is used. The average time taken to complete course-A without using the system is 80 sec and average time taken when using the system is 60 sec. This shows that there is 25% less time taken to do the same task when the system is used. Fig 4.7 shows course-B which is considerably more difficult than course-A to maneuver. This is similar to course-A Fig 4.5 where there is not much difference in total time taken but still our system works well. Fig 4.8 uses a proxy server to traverse course-B and we can see that the average time to traverse course-B is 18% less when we are using the feedback system than when not using the system. Total time taken to traverse course-A or course-B is less when using the feedback system.

The standard error rate of teleoperating is also high as there is one out of five chance that the participant would do something wrong and the experiment had to be started again. This can be due to network error or the participant is not patient enough to complete the whole experiment and does something wrong like crossing the white line in course-B. One of the limitations of the system is that there is a learning curve involved during navigating the bot with the help of video feedback. While panning left or right most of the time the operator would overshoot his required mark and has move to compensate that. This system is more useful if the video quality is lower as the system depends on the cpu to render the canvas If the quality of the video is high then more pixels have to processed which would slow the computer. The system works well if the lag is in between 1 sec to 7 sec anything lower would add additional overhead on the cpu and the performance of the system would degrade any anything higher the lag would increase exponentially.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this paper we proposed an algorithm which is suitable to teleoperate a bot from a remote location. We used HTML5 canvas to show the video than a standard video player. HTML5 canvas gives us more flexibility as we can manipulate individual pixels from the frame. The operator can pan left and right and move front and back. We devised two courses and four experiments to test our system. Our tests indicate that using our system takes less time to navigate than without using our system. We proposed a system that is suitable for teleoperating a robot located at remote location. Gives immediate feedback to the operater for smooth operation of the bot. Total time for doing the job is decreased using the feedback system.

The present system can be improved further by prioritizing moving objects in a video. For example if in a video only a specific object is moving and the background doesn't change, then we can only update that moving object before and get the background object later as it does change much. This would be useful in a mobile handset as it has limitations on the bandwidth available and the amount of space available on the system. This system would help in reducing bandwidth consumption and would also load faster.

REFERENCES

[1] Martin Hoernig, Andreas Bigontina, Bernd Radig, “A Comparative Evaluation of Current HTML5 Web Video Implementations”, Open Journal of Web Technologies (OJWT), 2014.

[2] Dilip Kumar Limbu, Wong Hong Vee Alvin, Chua Yuanwei, Albertus Hendrawan Adiwahono, Tran Anh Dung, and Han Boon Siew, “A Pilot Study on TeleBot, a Internet-based Teleoperated Robot”, 13th International Conference on Control, Automation and Systems (ICCAS 2013).

[3] Andrew Wichmann, Turgay Korkmaz, “Analysis of Deployment and Movement Policies in Wireless Sensor and Robot Networks”, World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE.

[4] Prima Dewi Purnamasari, Helen, Nadya Syifana, “Clickable and Interactive Video System Using HTML5”, Information Networking (ICOIN), 2014 International Conference.

[5] Yekaterina Kharitonova, Qiyam Tung, Alexander Danehy, Alon Efrat, Kobus Barnard, “Client-side Backprojection of Presentation Slides into Educational Video”, MM '2012 Proceedings of the 20th ACM international conference on Multimedia.

[6] Laurent Denoue, Scott Carter, Matthew Cooper, “Content-based Copy and Paste from Video Documents”, DocEng 2013.

[7] Manh Duong Phung, Thuan Hoang Tran, Thanh Van Thi Nguyen and Quang Vinh Tran, “Control of Internet-based Robot Systems Using Multi Transport Protocols”, 2012 International Conference on Control, Automation and Information Sciences (ICCAIS).

[8] Guolei Zhu, Fang Zhang, Wei Zhu, Yayu Zheng, “HTML5 Based Media Player for Real-Time Video Surveillance”, 2012 5th International Congress on Image and Signal Processing (CISP 2012).

[9] Keaton Mowery and Hovav Shacham, “Pixel Perfect: Fingerprinting Canvas in HTML5”, WEB 2.0 SECURITY & PRIVACY 2012.

[10] Günter Pomaska, “Presentation of a Virtual Architectural Model Considering New Multimedia Standards in HTML5”, Web3D '2012 Proceedings of the 17th International Conference on Web 3D Technology.

[11] Laurent Denoue, Scott Carter, Matthew Cooper, John Adcock, “Real-time Direct Manipulation of Screen-based Videos”, IUI'13 Companion, 2013.

[12] Sarah Osentoski · Benjamin Pitzer · Christopher Crick ·Graylin Jay · Shuonan Dong · Daniel Grollman · Halit Bener Suay · Odest Chadwicke Jenkins, “Remote Robotic Laboratories for Learning from Demonstration”, International Journal of Social Robotics 2012.

[13] Christopher Crick, Graylin Jay, Graylin Jay, “ROS and Rosbridge”, Proceeding HRI '2012 Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction.

[14] Christopher Crick, Graylin Jay, Sarah Osentosiki, Benjamin Pitzer,and Odest Chadwicke Jenkins, “Rosbridge: ROS for Non-ROS Users”, Proceedings of the 15th International Symposium on Robotics Research 2011.

[15] Andrew Wichmann, Burcu Demirelli Okkalioglu, Turgay Korkmaz, “The integration of mobile (tele) robotics and wireless sensor networks: A survey”, Computer Communications 2014.

[16] Laurent Denoue, Scott Carter, Matthew Cooper, “Video Text Retouch: Retouching Text in Videos with Direct Manipulation”, USER INTERFACE SOFTWARE AND TECHNOLOGY SYMPOSIUM 2014.

VITA

Sandeep Devarapalli

Candidate for the Degree of

Master of Science

Thesis: LIVE FEEDBACK ON A LIVE VIDEO USING HTML5 CANVAS

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the degree of Master of Science in Computer Science at Oklahoma State University in December, 2016.

Received the B.Tech degree from Jawaharlal Nehru Technological University Hyderabad, Telanagana, India 2013.