Fusion of Voice, Gesture, and Human-Computer Interface Controls for Remotely Operated Robot

Martin Urban and Peter Bajcsy
National Center for Supercomputing Applications (NCSA),
University of Illinois at Urbana-Champaign,
Champaign, Illinois, USA
{murban, pbajcsy}@ncsa.uiuc.edu

Abstract - *This paper presents an overview of a robot teleoperation system using voice, gesture, and human-computer interface (HCI) controls. The system consists of three basic software components including (a) acquisition and recognition of control commands from multiple inputs, (b) client-server network communication, and (c) command fusion and execution by a robot and its arm. The inputs for recognition of control commands come from (1) wired or wireless microphones, (2) wired orientation sensors mounted on human arms, and (3) HCI devices, such as a mouse, a keyboard or a text file with the sequence of control commands. The set of gesture commands is based on the US Navy lexicon for navigating aircrafts on the ground. Fusion of multiple commands is performed by (a) analyzing time delays and (b) assigning different priorities to commands and the clients issuing those commands. Consistent and conflicting commands are considered before a selected command is executed by a robot. For an emergency control, a video signal is sent to a monitoring station.*

Keywords: Robot control, command fusion, real-time navigation systems.

1 Introduction

In the past, several researchers have built systems that combine multiple sensors for autonomous vehicle navigation [7], [11]. In these applications, the problem of interest is about multi-sensor data fusion in order to reliably move a vehicle in an unknown environment. A fusion of higher level decisions [10], such as commands, is of interest when humans provide multiple inputs to a system. We are considering a related problem, where multiple human controls are used for navigating a robot in a hazardous environment, or for taxiing Unmanned Aerial Vehicles (UAVs) on an aircraft carrier deck in the presence of other manned aircrafts [7], [8], [9]. In both application scenarios, the control commands come from one of the directors in charge (e.g., hazard managers or flight deck directors) and should be executed by a specially equipped robot or by an UAV. To simulate the problem in a laboratory environment, we used an ActivMedia Pioneer 2DX robot as a surrogate and mapped the lexicon of flight director's commands to a set of motion instructions available to the robot. We explored multiple inputs for remote robot control including voice, gestures and human-computer interfaces (HCI). The overview of the system with multiple inputs is presented in Figure 1.

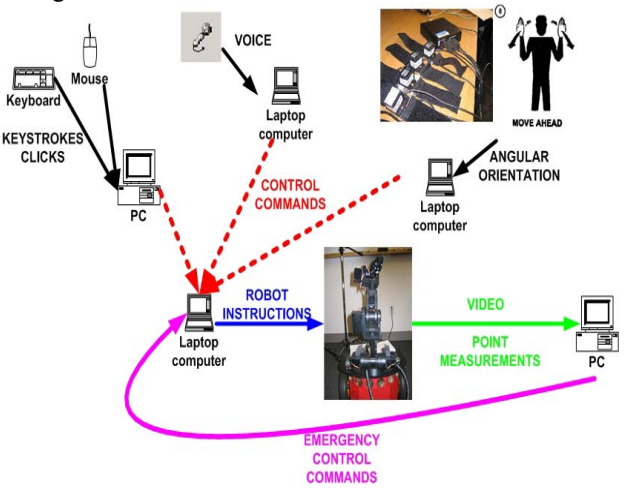

Figure 1: An overview of a system for remote robot control using sound, gesture and human-computer interface inputs.

First, we implemented robot control using HCI inputs. A user can use a keyboard and type in commands of his choice and their corresponding parameters. Second, we developed a template based speech recognition system so that typing can be replaced by more user friendly interface. Third, we added a gesture recognition system to accommodate remote control execution in very noisy environments, for instance, a carrier deck. Finally, we enabled robot arm control via mouse and keyboard interfaces in order to perform simple loading and unloading operations. For emergency control purposes, we mounted a wireless camera on the platform of a robot to obtain video feedback.

In terms of system architecture, the software is designed based on a client-server paradigm. All input devices (microphones, orientation sensors, keyboard and mouse) are attached to multiple computers that represent the

clients in the developed system. In our laboratory experiments the robot is connected to a laptop using the RS232 connection. This laptop acts as a server and accepts TCP client connections over the network. Each client can issue control commands to the robot by sending commands to the server laptop. The server fuses the commands from all clients, and resolves any conflicts that may occur. After command conflicts are resolved the commands are translated to a set of robot instructions. These instructions are sent to the robot via the RS232 connection, and are then executed.

The main objective of this paper is to describe the command processing flow as illustrated in Figure 2, and present the fusion of multiple robot controls. According to Figure 2, data are acquired first by using one of the previously mentioned controls. After the data are collected, they are classified using a recognition algorithm into one of the 21 possible taxiing commands (text and arm inputs do not require classification). Once the command is recognized, it is sent as a packet over a local area network (LAN) to a server that is directly connected to the robot. Next, the server fuses all incoming commands and decides which one will be sent to the robot. These commands are then translated to robot instructions which are executed by either the robot or its arm. Finally, a camera mounted on the robot provides video feedback that acts as another client controlling the robot. The server can handle multiple client connections at once and decides what commands should be executed.

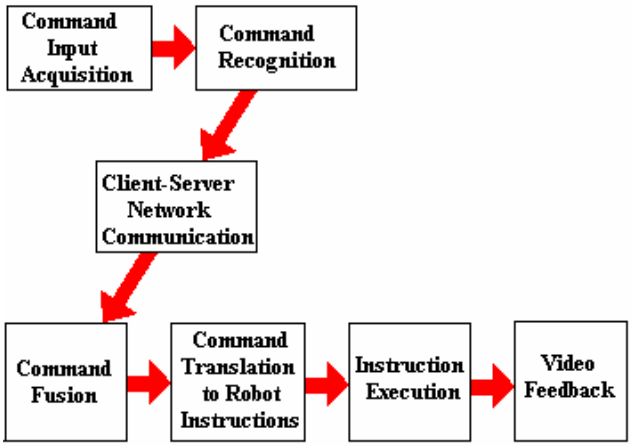

Figure 2: Robot control system flow. Command acquisition and recognition are described in Section 2. Network architecture is outlined in Section 3. Command fusion, instruction translation and execution, followed by video feedback are overviewed in Section 4.

2 Multi-Sensor Command Recognition

The developed system uses the client side to do all sensor input data acquisition and recognition. The recognition output is one of the commands defined by the US Navy lexicon [2]. The recognized command is sent to a server residing on the robot. There are two types of commands received by a server. First, it is the type of robot movement commands which control the direction and speed of the robot chassis. The second type corresponds to robot arm commands that manipulate an arm mounted on top of the chassis. We use four interfaces to control the robot, which are keyboard, mouse, microphone, and arm orientation sensors. The keyboard and mouse are also used to control the arm. In our experiments, we used Audio-Technica wireless microphones or a laptop mounted microphone as audio input, IS300 Pro Orientation Tracker sensors [1] as gesture input, and regular PC interfaces as HCI input. New inputs can be added without any change to the system, as long as any new client providing commands from another input device adheres to the communication conventions of our robot command network packets.

2.1 Commands from Human-Computer Interfaces

In general, commands from a client can control either robot movement or robot arm. In our application, human-computer interfaces are used for controlling robot movement or robot arm movement.

The simplest and most reliable method for controlling a robot and its arm is based on a keyboard input. In our current system, the keyboard input allows a user (a) to send single instructions or commands to the robot, or (b) to specify a file name containing a sequence of commands that is loaded and executed on the server side. If a command is sent then only a single command value is transmitted. When a single instruction is sent, then the server receives a string with up to three numerical values that are mapped to the ARIA ArRobot application programming interfaces (API) [4]. If a file name is sent then the server loads the text file from its hard drive and executes the sequence of instructions accordingly.

The Pioneer 2DX robot also comes with a 50cm robotic arm that has five degrees of freedom. The arm can be currently controlled via keyboard and mouse. One could implement a joystick control as well. The current implementation of the arm control allows users to move one arm joint at a time, or to move all joints at once to a set of predefined positions. Arm positions can be stored in memory, written to text files on the client side, and then re-used later on. We also implemented two transformations that covert 3D spatial coordinates (x,y,z) of the tip of the arm gripper to or from rotational angles of the joints. These coordinates represent the x, y, z location of the arm tip in millimeters from the origin defined at the center of the joint 1.

2.2 Commands from Audio Sensors

Our audio control using voice interface and based on microphone sensors utilizes (1) a lexicon of audio controls (temporally varying sound signals and their meaning), (2)

a definition of start and end of each audio control, and (3) an appropriate feature representation of all audio control signals that would maximize correct recognition. The lexicon of audio controls is usually defined by its control application. The temporal start and end points of an audio control can be defined by thresholding amplitude of the voice command from its background. Another approach is to use additional cues during voice recording, for example, a unique sound before the start and after the end of each audio command. The issue of appropriate feature representation can be resolved by (a) considering the number of distinct commands, (b) application specific real-time and computational requirements, (c) variability in repeating the same audio control, and (d) similarity of distinct audio controls.

In our approach, the audio control is executed in two phases. In the first phase also known as a training phase, samples (or templates) of all valid sound signals from a defined lexicon are collected, transformed into a set of features and stored prior to any recognition. The start and end points of each audio command are assigned manually, automatically or by well-controlled recording. In the second phase (at the run time), any incoming sound signal is parsed into candidate and background temporal segments by filtering and amplitude based thresholding. The candidate segments are transformed into a set of features and compared against the set of templates created beforehand. The comparisons measure the distance between the candidate sound and every template. The template with the lowest distance to the candidate determines the robot command. The temporally ordered sequence of interpreted robot commands is passed to the robot control component of the system. The drawback of the template based recognition method is that it is user dependent and requires each user to prerecord his set of templates.

2.2.1 Training: Template Creation

In order to train the audio recognition system, we first record at least one training sample in wave format for each word or phrase that represents a command. It is possible to record two or three samples per command to improve recognition accuracy but increase system's recognition time. These recordings are then checked for clarity of signal and manually (or semi-automatically) edited to remove beginning and ending silence and noise sections within the speech signal. After that, the recorded audio waves are converted to 8-Khz, 8-bit, mono format. The templates are formed by extracting audio features for each command. In our work, we used Linear Frequency Cepstral Coefficients (LFCC's) as audio features and they are defined in Equation (1). A new vector of P = 10 coefficients is generated for each K = 256 sample points of the wave at 128 sample intervals. The purpose of the conversion to 8-Khz is to minimize the number of extracted features, and hence decrease the run time needed for feature comparisons.

$$LFCC_i = \sum_{k=0}^{K-1} Y_k \cdot \cos\left(\frac{\pi \cdot i \cdot k}{K}\right) \quad (1)$$

2.2.2 Run Time: Speech Recognition

At the run time, a user says a command (a word or a phrase) into a microphone, which is recorded in a wave file format. The system then re-samples the wave to match the sampling frequency of the templates (8-Khz, 8 bit, mono). The re-sampled wave is filtered to detect the start and end points of speech segments. In the first part of the filtering process, the 4-th order high pass Chebyshev filter is applied to reduce the low frequency background noise. Next, filtering is completed by eliminating short length, high amplitude blips, glitches, and spikes (values are reduced to zero amplitude). The eliminated sounds correspond to noise characterized by less than 50 ms duration, with amplitude on the order of a spoken voice signal. We chose 50 ms since any recognizable speech sound a human makes is longer than 50 ms. Filtering is important to reduce unnecessary computation during silence periods and to preserve only speech signals in the command candidate wave. Similarly to the template creation described in Section 2.2.1, the LFCC features are extracted from the filtered wave. Finally, a DTW (Dynamic Time Warping) algorithm [3] is used to match the extracted features with all templates created previously. The template with the shortest error distance is selected to be the input command, unless the shortest distance is larger than a user-defined threshold of recognition. In this case, the input word is classified as unrecognized. This threshold could be experimentally estimated and it was set to 200 in our experiments. The DTW algorithm accounts for different temporal rates of speech. Figure 3 shows an illustration of DTW error computation for the input word "speech". The vertical axis is the template word and horizontal is the candidate word. $D_{i,j}$ is the overall error at times i and j for the chosen LFCC vectors in the two speech signals. It is calculated by summing the previous distance and the minimum local distance $d_{i,j}$ as shown in Eq. (2).

$$D_{i,j} = d_{i,j} + \min\left(D_{i-1,j}, D_{i-1,j-1}, D_{i,j-1}\right) \quad (2)$$

The distance $d_{i,j}$ in Eq. (2) is a Euclidean distance between corresponding template and candidate LFCC feature vectors. In this manner, the DTW-based algorithm finds the minimum global distance from the template and candidate speech beginnings to their end (bottom left to top right in Figure 3). The final error EDTW between two speech signals is the last computed $D_{i,j}$, where i and j are the final samples in their respective speech signals. EDTW corresponds to the upper right corner of the illustration in Figure 3.

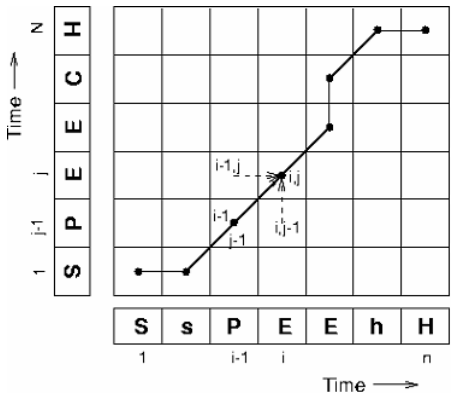

Figure 3 : Illustration of DTW algorithm used to compare two instances of the word "speech". The picture shows the shortest global path from beginning to end, as well as the calculation of error at coordinates (i, j).

2.2.3 Continuous Voice Signal Analysis

The voice recognition system has to be able to continuously analyze incoming audio. This is accomplished by recording the audio in 1 second long segments and each second analyzing the concatenation of the previous two recorded segments as shown in Figure 4. Thus, during each second, the previous two seconds of sound are analyzed. For example at time t=2 sections 1 and 2 are analyzed as shown in Figure 4. The reason for choosing a two second interval is because all the gestures in the US NAVY lexicon can be said using a normal rate of speech within this time limit. The reason for splitting the audio into one second segments is to account for the case when a command word starts in one segment but ends in the next as shown in first two cells of Figure 4.

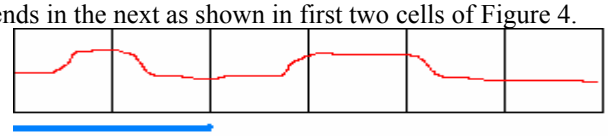

Figure 4: Continuous analysis of voice signals. Each cell represents 1 second. Centered red line represents silence, and raised red line represents a voice signal. Two one second segments are analyzed each second. The duration is illustrated by the blue bar.

It could occur in the described recognition system that a command is entirely contained within one segment of interval [t, t+1], and the intervals [t-1, t] and [t+1, t+2] contain only silence. In this case, this command will be recognized at times t+1 and t+2. We designed the system to disregard a command that was recognized in the previous second. Figure 5 illustrates a few critical timing cases. Silence is illustrated as relatively constant lines with middle range amplitude, and voice is represented by humps. The cases a, b, and c in Figure 5 will be recognized correctly. The case d will be classified as unrecognized because the real commands are recognized in the previous and following seconds. The case e will also be unrecognized because the real command occurred in the previous second. The case f cannot be recognized unless there is one second of silence before and after the shown segments, in which case it will be just like the case b. In the case g, the 2 seconds actually span 3 one second segments and one of the 1st or 3rd segments contains more than 50% silence. We found out experimentally that a word can be recognized correctly most of the time even if a small part of its signal is cut off at either end.

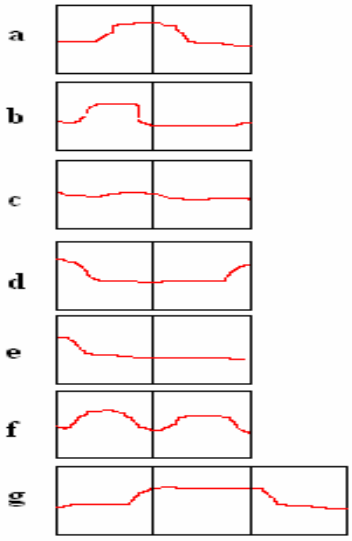

Figure 5: Possible cases of voice signals within the one second segments. They are: (a) one command spread across 2 seconds; (b) one command in one second and silence in the next; (c) silence; (d) end of command in first second and start of another command in 2nd second, (e) end of command, (f) 2 quick commands in succession, and (g) command spread across 3 segments.

2.2.4 Voice Recognition Success Rates

We achieved a 97.5% recognition rate using the described recognition method, and with a small vocabulary of two words, such as "go" and "stop". We used three user generated templates for each word, and 20 recordings to be classified for each word. The run time for classifying each word was less than 0.5 seconds.

2.3 Commands from Arm Orientation Sensors

In addition to the previously described robot control interfaces, we developed an input control interface based on arm gestures. The arm gesture recognition system uses four orientation sensors [1] attached to person's arms. Each sensor reports the yaw, pitch and roll values (Euler angles). The values from all sensors are analyzed by modeling gesture commands in the US NAVY lexicon [2]. Once a gesture is recognized a packet containing the appropriate gesture value is sent to a server. One of the

differences between the gesture recognition and voice recognition is that the voice command is said only once, whereas the gestures are frequently formed by periodic arm motions. If a gesture is recognized then theoretically one should be sending packets containing a command continuously. We decided to send a packet only the first time a gesture is recognized and another packet when that gesture stops being recognized. The gesture recognition system is described more extensively in our previous work [5], [6].

2.3.1 Gesture Recognition Algorithm

The algorithm receives three values (yaw, pitch and roll angles) from four sensors mounted on two arms. Each value is classified into steady or oscillating. Next, the value is categorized based on the magnitude as high, medium or low. Using the above classes and categories, each of the gestures in the lexicon was modeled so that the recognition could take place in real-time. The drawback of gesture recognition approach is that any new addition to the already established lexicon of commands requires developing new gesture models.

3 Client-Server Architecture

3.1 Client-Server Paradigm

The server and clients communicate over a network using TCP. Most of the communication comes from the clients, but the server can also send packets back to the clients. When a client recognizes a new command from its input source it will send a packet to the server. There are four types of packets that are used. Robot command packets contain a numerical value that is bound to a file containing a set of instructions on the server that the robot will execute. Arm packets contain values representing the current and desired, positions, movement velocities and other parameters associated with all the joints of the robot arm. File name packets contain a string of a file that is to be loaded, but was not bound to a numerical robot command beforehand. Finally, single instruction packet types send across a value representing one instruction as well as the parameters that this instruction requires. Each packet is marked with a byte representing its type and the priority of the client sending it.

The arm packet type always contains six values representing each joint of the arm. There is also a byte which determines in what context these six values are to be used. Arm packets going to the server can only be used to tell the arm joints to move to the positions specified. However, arm packets coming from the server can be of several types. Most likely they are just update packets telling the clients the current position of the arm joints, but can also represent the minimum, maximum, center, home, and movement velocity values of the joints. These are necessary for the client to perform sanity checks on allowed arm movements as well as some calculations like converting arm joint byte values to degrees and vice-versa.

The robot type packet can also vary. All of our robot type packets come from the clients, and we never send any robot type packets from the server, although this can be done. Most of the time the packet contains a single command value, which is mapped to one of the NAVY lexicon gestures. Such packet tells the server to execute a file containing instructions representing this gesture. There can also be other flags attached that can immediately stop the arm or robot without having to be deciphered as commands.

File name packets and single instruction packets have a non varying structure and always contain the same kind of information. Both are to be used for client to server communication only.

3.2 System Advantages and Disadvantages

The advantages of this client-server architecture are modulation, no human presence required at the location of the robot, and wireless remote control of the robot. Each client can be written separately and in a different programming language. It can also run on any operating system as long as the client can send and receive a byte stream over a TCP connection. This does not require for the server or other clients to be changed whenever a new client is created. This is very useful for the addition of new command interfaces in the future as the existing structure does not need to be changed.

There are several disadvantages to this system however. First are the delays that result from the network, these appear to be insignificant for us but if the robot is to be controlled across continents they might present problems. The second drawback is that if the users on different clients have conflicting agendas they can hinder and undo each others work, which requires an implementation of client priorities. However we assume the clients have a common interest.

3.3 Command Delays

There are numerous delays that happen between the time a user issues a command, and the time that the robot starts performing the instructions associated with the given command. These delays are shown in Figure 6. Some delays occur on the client side, while other delays occur on the server side. The server side delays are independent of the input method used on the client side. Therefore the server side delays have the same effect on all commands.

The first client side delay is caused by the acquisition of the input. In the case of speech recognition, it can take up to 2 seconds to issue an audio command. For gesture recognition the time to repeat an arm movement enough times to detect oscillation can also take a couple of seconds. The next client side delay is caused by the recognition algorithm. In speech and gesture cases, recognition algorithms take less than 500ms. However the

recognition delay is also dependent on the client machine speed. Since the purpose of our system is to control the robot in real time we assume that all clients can handle the command recognition in a small amount of time.

The next delay results from the command being sent over the network to the computer running the robot server. This delay can vary with the current performance of the network and internet, but under ideal conditions, all the clients are located on the same network as the server. If we assume that everything is on the same network the delay is less than 1 ms.

After the packet is sent over the network to the server, three types of delays occur on the server side. The first delay comes from the analysis of command conflicts and the command to robot instruction translation. Command conflicts are resolved in an insignificant amount of time. The command to instruction translations can take up to 0.3 ms if a direct hard drive access is used. However, if we loaded all the command to instruction translation files into memory when the server starts, the translation delay could be decreased to about 0.001 ms. Next server side delay is caused by sending the instructions to the robot over an RS232 serial cable, but we have no way of measuring how long this takes. Finally, there is also a delay that occurs on the robot itself when it sends the appropriate wheel velocity commands to its motors, as well as the delay of overcoming the inertia of its previous state.

If we disregard the sensor data collection time, the overall biggest delay occurs in the client side recognition algorithm. This can cause problems if one client is faster than another, or if the recognition algorithm is less complex for one type of input than another. Therefore it is possible that one command is issued first on a slower client followed by another command issued on a faster client. Due to the recognition processing delay the command that is issued first is actually sent over the network after the command from the faster client, and will also be executed second by the server.

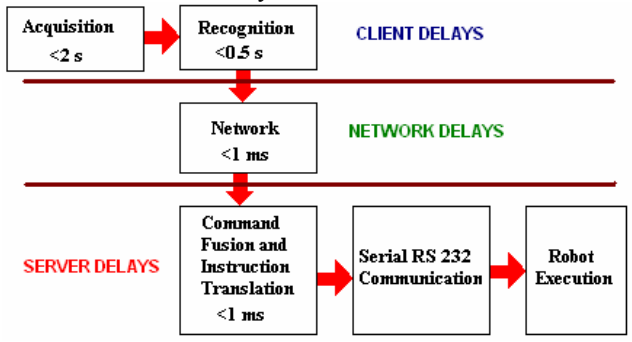

Figure 6: Delays occurring between the time a command is given and executed by robot

4 Fusion and Execution of Commands

When more than one command is received by the server at almost the same time command conflicts can occur. By our default, a conflict can occur when a new command is received within 250 ms after the previously received command. To deal with conflicts we introduced two types of priorities, such as command and client priorities. Command priorities decide which possible commands are more important than others. Client priorities correspond to the reliabilities of the clients' command recognitions. Another way of dealing with the command conflicts is to use a majority voting method. We assume that all the clients trying to run the robot have the same intentions and are monitoring the same environment with different sensing methods. Thus, when many commands arrive at the same time, the command with the highest count will be executed once and the others will be discarded. Currently we check command priorities first and if the conflict has not been resolved then we check the client priorities. The reason why client priorities are not checked before command priorities is to allow stop commands to have the highest priority no matter what client type they come from.

We describe assignment of priorities in the next two sub-sections. Then, we classify the simultaneous commands into consistent and conflicting commands, and describe them in sections 4.3 and 4.4. One could approach the problem of simultaneous occurrences of multiple commands by combining the commands (e.g., Move Ahead and Turn Left). However, this solution would have to be driven by other applications and the command combination would have to be carefyllu scrutinized, for instance, Move Ahead and Stop.

4.1 Command Priorities

To cope with command conflicts we prioritized the 21 possible NAVY commands. When the second received command had higher priority than the first one, the second one would overwrite the first command. If the second command had equal or lower priority as the first one, then the first one would continue its execution until it finishes or another command overwrites it later in time. All commands were given priority value ranging from 0 (highest priority) to 21(lowest priority). Due to the safety issues related to the robot and its surrounding, the Stop command was given the highest priority. We decided to rank the priority of the other remaining commands based on the robot movement velocity. Thus, the next highest priority after Stop was given to the commands Brakes, then Slow Down, and Slow Down Left/Right Engine. The commands with constant velocities, such as Move Ahead or Turn Right/Left were given equal priorities, but lower than the previous group. The next lower priority was given to commands that increase the movement speed of the robot. An unrecognized command was given the lowest priority.

4.2 Client Priorities

Each client has a different recognition success rate and therefore commands coming from some clients are more reliable than those coming from other clients. When two or more conflicting commands are received from different

clients at about the same time it may be hard to decide the order of execution or perhaps which commands should get executed and which ones should not. For this reason we introduced a client priority value. This way clients with high reliability, such as pure text and mouse input clients have the highest priority. We assume that a user always inputs correct commands. Clients that involve gesture, speech or other sensors with recognition rates below 100% have their priorities reduced based on our experimental results of their success rates. One problem that arises with the client priorities is whether the priority value is assigned at the client or server end. The server does not know what kind of clients are connected to it, nor can it know beforehand what new clients types will be connected in the future. The design of our system is to accommodate an unlimited number of various client types and hence the priority value is decided on by the client itself. This priority assignment could present a problem if an unreliable client would assign itself the highest priority. To remedy this potential problem, we use a filter on the server side that accepts or rejects client connections based on their IP addresses or other criteria. In this manner, only clients that are trusted by the server and correctly assigned a priority value for themselves are accepted.

4.3 Consistent Commands

The first class of simultaneous commands is labeled as consistent commands. Simultaneous consistent commands occur when each client recognizes and issues the same command. Figure 7a shows two commands of type K being received within the 250ms conflict detection interval. If command K (e.g. Move Ahead) arrives at the server at time t, and again at time t+dt (dt is less than the default priority delay check of 250 ms), only the first copy of the command K issued at time t is executed. There is no need for the server to resend the same command to the robot again since one of the two situations would happen. First, the robot's actions would seem unaffected, for example, the robot is already moving ahead. Second, the robot might repeat what it has already done or started doing at the first execution of the command K, for instance, it is turning at a given angle. For this example, the robot could end up in an undesired position because it would turn twice the desired angle. As a consequence, time would be wasted to issue a new command to correct the robot's position. The priority delay check value is insignificant compared to the time it takes for the robot to move to a new position. Thus, by the time all clients' sensors detect and recognize a new command, the delay timer on the previous command will have expired. For an expired delay timer, new incoming commands will not be in conflict with the commands previously executed between times t and d+dt.

4.4 Conflicting Commands

The second class of simultaneous commands is labeled as conflicting commands. As shown in Figure 7b, conflicting commands occur when a command K is issued at the time t and a command L at the time t+dt. If L has higher priority, then L will overwrite K starting at time t+dt. This would be the case of K being Move Ahead, and L being Stop. However, if L has the same priority as K (e.g. Turn Left and Turn Right) then L will be ignored and K will continue to execute. If L has lower priority than K, then again L will be ignored and K will continue to execute. The third possibility is that more than two commands are received between the times t and t+250 ms as shown in Figure 7c. In this case, the majority voting method is used. The count of each command type received is determined and the one with the highest count wins. In the case of two command type counts being equal, we consider again the priorities of all commands as well as the average or maximum priorities of the clients sending each command. For example if 10 commands arrive and 9 are the same, then the most likely command is the one that arrived 9 times. However, if 10 commands arrive with 4 being of type K, 4 of type L, 1 of type M and 1 of type N, as shown in Figure 7c then the answer is not as clear. In this case, K and L both have the highest occurrence. Thus, the command priorities of K and L are compared, and if the conflict were not resolved then the client priorities would be used. We could either take the average priority of the 4 clients of K and the 4 clients of L, or use the highest client priority from each to determine which one from K or L would be executed as described earlier in this section.

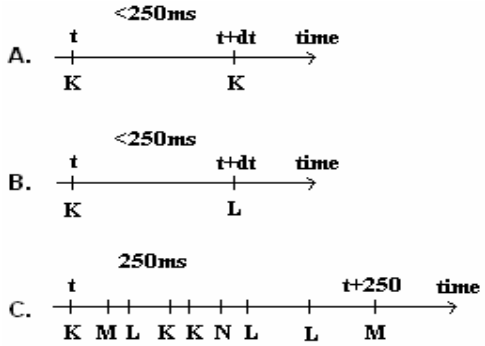

Figure 7: Illustration of possible command conflicts. The symbols K, L, M and N denote different robot commands. The pictures show the cases of A) 2 identical commands, B) 2 conflicting commands, and C) many conflicting commands.

4.5 Video Feedback

A forward facing camera mounted on the back of the robot provides visual feedback about the command execution. This feedback is analyzed by another client, which then sends robot commands to the server. The video

from the robot's camera can be analyzed for various reasons. Currently we use it for movement and color detection.

The motion detector analyzes the scene of the video and is attuned to find human controllers. It is intended to detect when a new arm gesture operator attempts to control the robot. We use operators wearing a red and yellow shirt. When one operator intends to pass control to another one he gives a "Pass Control" command by pointing in the direction of another human controller. In order to execute the "Pass Control" command, the robot needs to find the new operator, which can only be done through a visual search. Video analysis can provide the direction in which the new operator is located. The video feedback will then send a turn robot command to the server. The video is continuously analyzed to detect the correct shirt color until the robot is turned in the direction of the new operator. At this point, a new operator and its client controls take over the robot operation. The old operator's client can be either disconnected or ignored as it is no longer in control of this robot.

5 Conclusions

We have developed a system that can be used to control a robot using an unlimited number of clients and address the problem of command control fusion. In our experiments we used four input types to control the robot and the arm. These were human-computer interfaces such as keyboard and mouse, as well as other interfaces including an audio sensor and arm orientation sensors. The control of the robot can also be handled by sensor input types not discussed in this paper. The prototype robot control system is independent of the types of clients that connect to the robot. The main focus of our paper is on the fusion of all input methods into one robot command. We considered timing delays, simultaneous commands, and client and server interpretation of robot commands. Video feedback from the robot was used as another control client to improve system's performance when multiple robot operators are present.

The developed system does not necessarily have to be used for simulations of UAV navigation. It can be used for any user-defined lexicon of commands, and a user can create his command to robot instruction mappings. Similarly, the speech recognition software can be used by itself, since it is template-based, and the vocabulary of valid commands can defined depending on an end application.

6 Acknowledgement

This material is based upon work partially supported by the National Center for Advanced Secure Systems Research (NCASSR) of the Office of Naval Research (ONR).

References

[1] Intersense web site: http://www.intersense.com/

[2] US Navy, "Field Manual FM1-564 Appendix A", web site: http://www.adtdl.army.mil/cgi-bin/atdl.dll/fm/1-564/AA.HTM

[3] Wrigley Stuart N. "Speech Recognition by Dynamic Time Warping", web site: http://www.dcs.shef.ac.uk/~stu/com326/

[4] ActivMedia documentation for ARIA, ActivMedia support web site: http://robots.activmedia.com

[5] Urban M., P. Bajcsy, R. Kooper and J-C Lementec, "Recognition of Arm Gestures Using Multiple Orientation Sensors: Repeatability Assessment," the 7th International IEEE Conference on Intelligent Transportation Systems, Washington, D.C., October 3-6, 2004, pp 553-558.

[6] Lementec J-C. and P. Bajcsy, "Recognition of Arm Gestures Using Multiple Orientation Sensors: Gesture Classification," the 7th International IEEE Conference on Intelligent Transportation Systems, Washington, D.C., October 3-6, 2004, pp 965-970.

[7] Kak A. and A. Kosaka, "Multisensor Fusion for Sensory Intelligence in Robotics," Proceedings of Workshop on Foundation of Information/Decision Fusion: Applications to Engineering Problems, August 7-9, 1996, Washington D.C.

[8] Cheng, V. H. L., V. Sharma, and D. C. Foyle, "Study of aircraft taxi performance for enhancing airport surface traffic control," *IEEE Trans. Intelligent Transportation Systems*, Vol. 2, No. 2, June 2001.

[9] Taxiway Navigation And Situation Awareness System (T-NASA) web page: http://human-factors.arc.nasa.gov/ihi/hcsl/T-NASA.html.

[10] Fong T., "Collaborative Control: A Robot-Centric Model for Vehicle Teleoperation," Ph.D. Dissertation, CMU-RI-TR-01-34, November 2001 (156 p).

[11] Kam, X. Zhu, and P. Kalata. "Sensor fusion for mobile robot navigation." Proc. IEEE, 85(1):108--119, Jan. 1997.

[12] Luo, R.C. Lin, M.-H. Scherp, R.S. "Dynamic multi-sensor data fusion system for intelligent robots," IEEE Journal of Robotics and Automation, Vol. 4, No. 4 August 1988.