Standardized Linearization and Vectorization Algorithm for Arm Motion Control of A Humanoid Telepresence Robot

Michael Waddell, Joel Villasuso, Daniela ChavezGuevera, and Jong-Hoon Kim
Discovery Lab, School of Computing and Information Sciences
Florida International University, Miami, FL 33199 USA
kimj@cis.fiu.edu

Abstract-- **Recently, the new motion sensor: the Kinect is being used for natural motion retrieval with no additional equipment on the operator, less computational demand, and it is cost effective. But many restrictions apply because the results retrieved from the Kinect include noisy data which disturbs precise and smooth robot motion control. We propose a simple linearization algorithm to improve the accuracy of the data retrieved from the Kinect and designed a vectorization algorithm for converting positions of an operator's skeleton in three dimensions to robot motions. In this paper, we provide the algorithm and its implementation. Furthermore, we demonstrate the performance of the algorithm using the prototype robot.**

***Index Terms*--Kinect, Motion Sensor, Humanoid Telepresence Robot**

I. Introduction

Tele-presence robots are capable of allowing user interaction with an environment remotely. This concept is used in different applications such as navigation, tele-surgery, sub-sea work, education, and outer-space exploration. Some tele-presence robots are built to physically represent their human operators when interacting with other individuals in remote areas. Others are intended to be an avatar in dangerous environments which require effective human to robot interaction. Well known tele-presence robots such as QB [1] and Navigoid [2] are able to physically represent their human operators but they lack key human-like gestures and emotions. The Telebot [7,8] is a project that uses the concept of tele-presence to physically enable handicapped military veterans and police officers to return to their public duties by creating a humanoid robot that is easily and accurately controlled by the officer [3]. The Telebots system includes a balancing mechanism, head, vision, hand, and arm control. Arms are the Telebots primary means of manipulating its environment, and their control is the focus of this research. They allow the Telebot to defend itself, interact with humans by means of a handshake or a hug, and even communicate using sign-language.

Accurate arm motions are therefore a vital component of the overall function of any tele-robot. This paper presents a simple Vectorization Algorithm for interpreting and then emulating a person's arm movement, and a Linearization Algorithm to improve the accuracy of the data retrieved from a Kinect sensor. The goal of this research is to provide a simple way of using these algorithms to control robotic arms that can easily be reproduced for any purpose and for any combination of sensor hardware and computing software.

This paper is organized by first explaining our methodology, then its implementation, and then finally an evaluation of the results.

II. States of Art

Various groups have attempted to create solutions to accurately translate human arm gestures in a robot using the Kinect sensor. Such works include "Real-time 3D Object Tracking Using Kinect Sensor" [4] by Takayuki Nakamura which uses complex point recognition tracking algorithm to determine the changing coordinates of an arm in 3D space, and "Design and Implementation of Human-Robot Interactive Demonstration System Based on Kinect" [5], by Liying Cheng et al, which uses a simple product of vectors to determine the angles between different vectors and does not use any filters to adjust for noisy data. The two major faults of the current research done on arm control is that the Vectorization techniques are far too specialized and hard to replicate, while the Linearization techniques are lacking in their ability to dynamically account of sensor noise. In "Full-Body imitation of human motions with kinect and heterogeneous kinematic structure of humanoid robot" [5], Van Voung Nguyen attempts to map the obtained kinematic data from the Kinect to control a humanoid robot, but it falls short in implementing it beyond a computer simulation. In "Developing a Gesture Based Remote Human-Robot Interaction System Using Kinect" [6], Qian, et al. succeed in controlling a robotic arm using the Kinect sensor, however it is limited in that its primary function is to recognize and imitate a limited library of gestures.

We address these issues by creating easily replicated and broadly applicable Vectorization and Linearization Algorithms.

III. Proposed Algorithm

The Vectorization Algorithm (VA) translates new data from the optical/depth sensor into usable data. To present the VA, we will focus on the angle Alpha in the YZ-Plane.

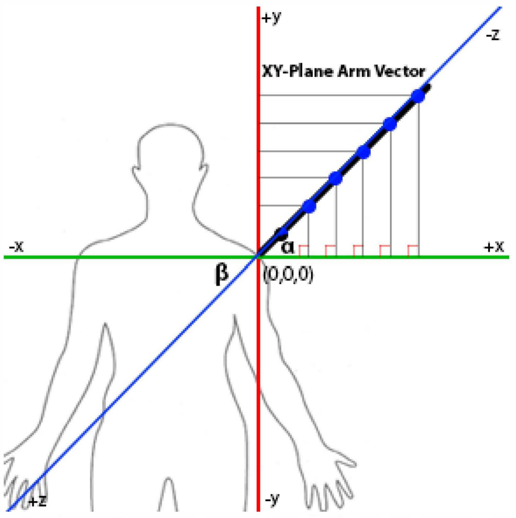

Figure 1: X,Y, and Z Shoulder Coordinate System

A. Vectorization

The VA is the method we use to calculate the angles Alpha, Beta, Gamma, and Delta and it is significant because it quantifies the 3D movements of human arms in vector space. A 3D environment is created and any movement that the optical/depth sensor measures was interpreted as changes of angle Alpha between the vectors in Fig. 1 X-, Y-, and Z-coordinates. The arm vector, which has a fixed magnitude, is created using the shoulder point as a fixed XYZ origin (0,0,0), and the dynamic X-, Y-, and Z-coordinates of the elbow point as it changes in 3D space and is given by the equation:

$$(XYZ\_Elbow - Point) - (XYZ\_Shoulder - Point)$$

The arm vectors 3D movements are represented by vector projections in two, 2D planes, one which represents its movements in the YZ-plane and the other, the XY-plane. The arms vector created in the YZ-plane is given by the equation:

$$(XY\_Elbow - Point)(XY\_Shoulder - Point)$$

Angle Alpha in the YZ-plane combines with angle Beta in the XY-Plane created the arms 3D movement. Each angle is calculated using the Law of Cosines, and in 2D space these triangles are right triangles created by the dynamic YZ- and XY-vectors and the fixed virtual vector pinned to the shoulder point, as shown in Fig. 1. The X-axis is not plotted in the YZ-plane, and therefore any change in its value is represented as a change in projected magnitude of the YZ arm vector.

Since Alpha is calculated with right triangles, any change in the magnitude of the YZ arm vector as a result of solitary movement in the X-axis creates similar right triangles and therefore Alpha does not change. This is how the two planes are separated and their respective angles calculated using translated local coordinates. To further adjust for rotation that might otherwise decrease the accuracy of the angles, Rotate(X, Y, and Z) is used to calculate changes in the body's rotation relative to the global coordinate system's XY- and YZ- planes. Once an offset rotation angle is calculated to be of a certain "x" degrees using quaternion calculations, the rotation of the joint is compensated and eliminated by an equal and opposite "-x" degrees. In this way, as far as the code is concerned the body and its joints' rotation do not factor into the calculations of Alpha, Beta, Gamma and Delta. Angles Gamma and Delta use the exact same method to combine two, 2D planes to create 3D movement of the elbow joint, as shown in Fig. 2.

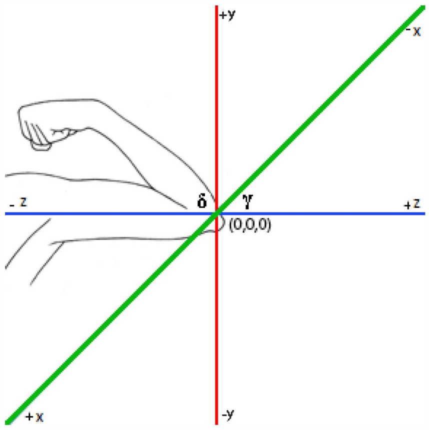

Figure 2: X,Y, and Z Elbow Coordinate System

There are two differences with the elbows servos. The first is that they are rotated at a 90 degree angle so that the elbows first servo (Servo-3, controlled by angle Gamma), which provides rotation, moves in the XZ-plane while the second servo (Servo-4 which controls angle Delta) moves in the YZ-plane. The second difference is that since the elbows rotation is actually physically provided by the shoulder, Servo- 3 is instead added as the 3rd servo on the shoulder joint, as shown in Fig. 3. While our VA is accurate, sensor noise obscures the true values of the angles, and one way of dealing with this through filtering the data through a Linearization Algorithm.

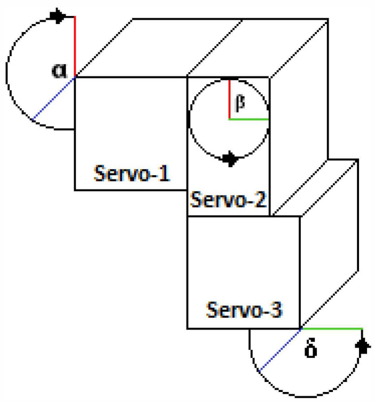

Figure 3: X,Y, and Z Servo Coordinate System

B. Linearization

Limiting noise is possible by using filter algorithms that calculate measured values more representative of their true values. Many robotics software use Non-Linear

Filter Algorithms, however these are limited because their filtering algorithm does not change dynamically. A Linear Filter Algorithm known as the Kalman Filter is extremely effective by comparison and our simplified Linearization Kalman Filter (LKF) is explained below in terms of the angle Alpha. The LKF works in two recursive steps that occur during each calculation of Alpha. The first step adjusts for magnitude and either adds or subtracts the dynamic Expected Measurement Variance (ExpM) from the measured value (M). The ExpM is the magnitude of the measurement error we expect to get from the optical/depth sensor, and M is the angle Alpha resulting from the VA. The ExpM is either added or subtracted depending on whether or not the arm is moving within certain minimum and maximum sensor thresholds that need to be accounted for. The second step adjusts for measurement noise by transforming the new angle Alpha from the first step by comparing the measured value against an expected value using the following equations:

$$Weight = \frac{ExpectedError}{ExpectedError + ExpM}$$

Where Weight is a value between 0 and 1 that determines if the expected error of the expected value (ExpE) is more or less significant than ExpM. The Weight will approach if ExpE is larger than ExpM, and it will approach 0 if ExpM is larger than ExpE, affecting the new Alpha in the following way:

Alpha = [(1 -Weight) * (E)] + (Weight) * (M)

If ExpM is larger than ExpE, then we expect the measured value of Alpha to be less accurate than the expected values of Alpha, so we give them less weight when calculating the new Alpha. The new Alpha will be a sum of E added to M with each value adjusted by the Weight. The final step is to update these values for each iteration by creating a new E by adding a direction and velocity metric to the previous value of the new Alpha, and testing this against the next M in the same process outline above. The direction metric is determined by calculating whether the values of E are increasing or decreasing between each iteration, and the velocity metric is determined by the magnitude of this change.

IV. Implementation

Our vectorization and linearization algorithm are implemented by Java with OpenNI library and XBox Kinect Sensor which provides 20 joint points of a body skeleton in 3D space.

A. Implementation of Vectorization

The VA sample code segment using the angles, Alpha and Beta, which control Servos -1 and -2 for one arm is follow:

Section : 1

- PVector axis = new PVector(0, 100);
- PVector axis = new PVector(0, 100);
- PVector armYZ = new PVector(y , z);
- PVector armXY = new PVector(x , y);

Section : 2

- PVector axis = new PVector(0, 100);
- PVector axis = new PVector(0, 100);
- PVector armYZ = new PVector(y , z);
- PVector armXY = new PVector(x , y);

Section : 3

- Alpha = map(Alpha, alphaMax, alphaMin, alpha2Max,alpha2Min);
- Beta = map(Beta, beta2Max, beta2Min, beta2Max,beta2Min);

Implementing the VA begins in Section 1 with the creation of three PVectors: axis, arm yz, and arm xy. PVector axis creates a stationary vector by which the arm vectors in the YZPlane and XY-Plane will be compared. PVector arm yz creates the users arm vector movements in the YZ-Plane. PVector arm xy creates the users arm vector movements in the XYPlane.

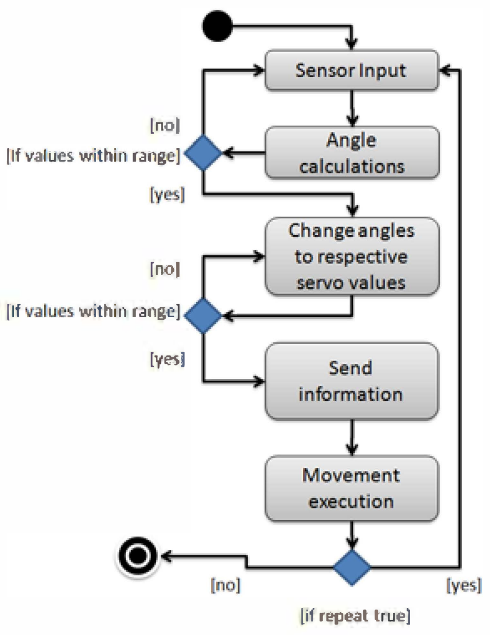

Figure 4: Logic Flowchart of Code

Section 2 creates the angles Alpha and Beta by using the function angleBetween() which calculates angles between the dynamic PVectors arm yz and arm xy and the fixed PVector axis using the Law of Cosines. Section 3 transforms the angles Alpha and Beta into values that will be sent to Servo-1 and Servo-2 using the map() function which creates a ratio of Alpha and Betas minimum and maximum values (defined by alpha min, alpha max, and beta min, beta max) and Servo-1 and Servo-2s minimum and maximum values (defined by alpha 2 min, alpha 2 max, and beta 2 min, beta 2 max). The values of alpha 2 and beta 2 are then sent to Servo-1 and Servo-2 respectively. Servo-1 controls the vertical (YZ) movement while Servo-2 controls the horizontal (XY) movement. The angles Alpha and Beta are sent as a string in the format as below.

< servo id + angle + speed >

These inputs are sent through a serial port to the processor that communicates with the Servos independently. The flow of the logic of this entire process can be seen in the Flowchart of Fig. 4.

A sample of 600 iterations of the angle Alpha can be found in Fig. 5 and is meant to represent a change in the X-axis in the YZ-plane where Alpha should not significantly change. It is obvious that the data collected from the VA suffers from a significant amount of noise that obscures its true values. We can solve this through the implementation of the LKF.

B. Implementation of Linearization

Implementing the LKF code begins in Section 4 by assigning the value of expected value of Alpha (Exp) as the function KFilter. The function first determines if the measured value of Alpha (M) is within the minimum detectable threshold of the Kinect (min detect, which is around 25-30 degrees); a lower bound where the Kinect overcompensates by adding its own expected measurement error. If M is below this threshold, it is ExpM higher than it should be; if M is above this threshold, it is ExpM lower than it should be. In Section 5, we see the exact same arithmetic previously outlined. The new value of Alpha, R is returned and the function is completed for this iteration. Exp is updated as the value of this returned R. In Section 6, Exp is transformed using the direction and velocity metrics, which determine the direction the arm is moving in by analyzing if each new iteration of Exp is greater or less than the previous value. For instance, if the newly updated Exp is greater than its previous value (prev), then the velocity metric is added because we expect the next value to continue to increase. The magnitude of the velocity vector is determined a moving average of the magnitude of change at each step.

Section : 4
- Exp = KFilter(Exp, M, Err, ExpM);
- float KFilterAlpha(float Exp, float M, float Err, float ExpM);
- if (M > minDetect) M = M + Exp.M;
- if (M < minDetect) M = M - Exp.M;

Section : 5
- W = (Err)/(Err+ExpM);
- R = ((1 - W) * Exp) + (W * M);
- return R ;

Section : 6
- if (prev ¡ E) f
- DiffAlpha = Exp - prev;
- StoredDE = DEAlpha;
- Exp = (Exp + DEAlpha);

Section : 7
- AlphaZ = map(Exp, alphaMax, alphaMin, alpha2Max, alpha2Min) ;

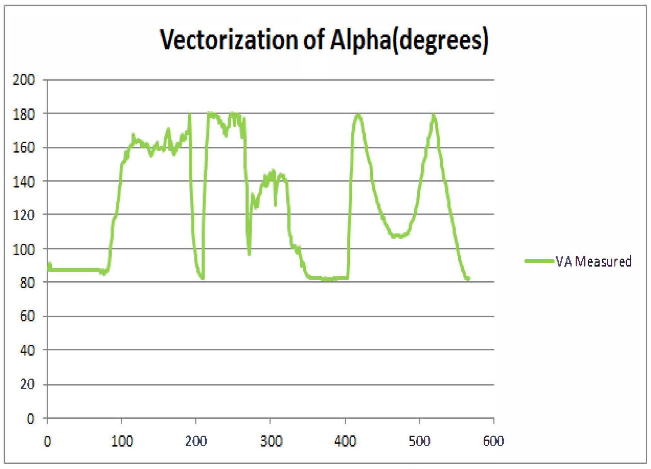

Figure 5: Vectorization Applied to Alpha

This process modifies the value of the angle Alpha sent to Servo-1 because the Alpha we obtained by the VA has now been put through the LKF, changing the previous code to Fig. 5. A sample transformation by the LKF on the 600 iterations of the angle Alpha previously measured by the VA can be seen in Section 7, which shows the rigid troughs and peaks as lessened in the graphical representation of the users arm movement in the YZ-Plane and what we would expect to see in a solitary movement in the X-axis.

V. Evaluation

Evaluating the validity of the VA and its implementation is by seeing how the calculated values of the Alpha using the VA compare to the absolute values of Alpha sent to Servo-1, as shown in Fig. 6. The Alpha value sent to Servo-1 changed from 0 degrees to 180 degrees twice over an interval of 800 iterations in Fig. 6.

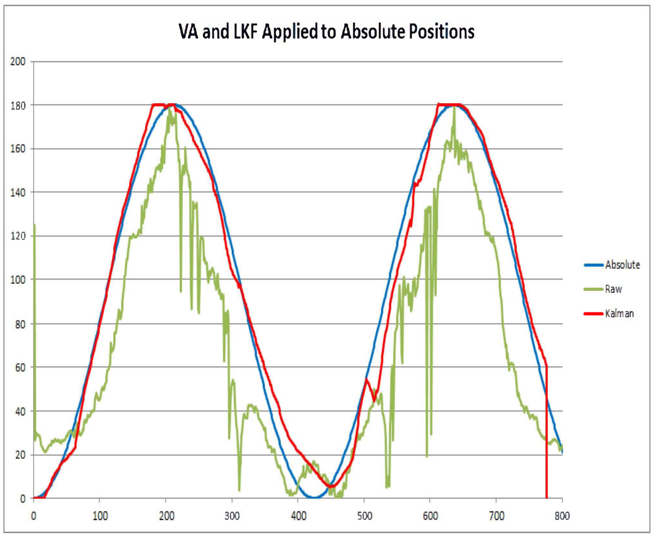

Figure 6: Comparison between Absolute, Raw, and Filtered Data

These values are known values that have zero noise since they are sent directly to Servo-1 as test values of the angle Alpha, representing the true movement and therefore absolute position of Servo-1. The Kinect then read and calculated these values using the VA. The values are plotted versus their Absolute values, as shown in Fig. 6 as the Raw Series. While the VA accomplished its task of calculating and then emulating the absolute value of the angle Alpha, there is an obvious variance present in the form of both noise and an incorrect magnitude. One possible way of mitigating this noise is through the LKF, which allows for the obscured values of Alpha to be transformed into a smooth graph that more closely resembles the Absolute movement, as can be seen in Fig. 6 as the Kalman Series. The LKF has done a great job in reducing the amount of noise that significantly reduced the accuracy of Alphas value. Using an integral function to determine the accuracy of both the Raw and the Kalman series by comparing them to the Absolute graph, on average the Raw measured VA values had a 73.6 percent accuracy, while the LKF values had a 96.3 percent accuracy. By first calculating the angle Alpha via the VA, and then putting it through the LKF, Servo-1 will move in such a way that it appears to be more accurately emulating the users arm movements. This exact same method is used with the angles Beta, Gamma, and Delta that are sent to Servo-2, Servo-3 and Servo-4.

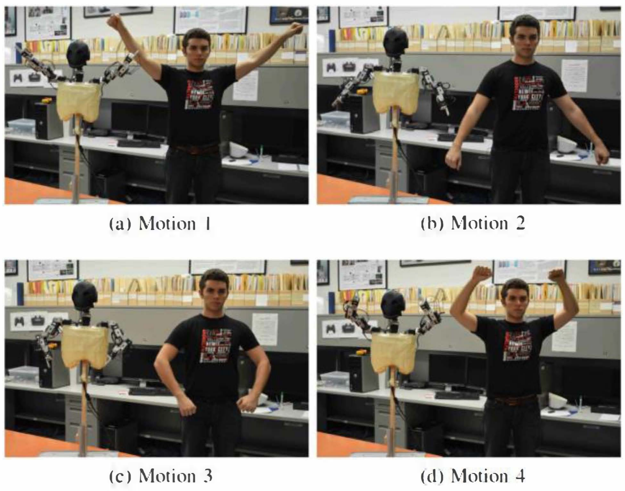

Figure 7: Motion Control Implementation Demo

Our vectorization and linearization algorithm can provide proper angles of operator's each joint into a robot controller so that our test robot nicely imitates operator's motions as depicted on Fig. 7.

VI. CONCLUSION

The VA has limitations in its calculating ability at certain thresholds when trying to calculate the angles Alpha, Beta, Gamma, and Delta. At these thresholds that exist around 0 degrees and 180 degrees, the noise from our optical and depth sensor reaches its maximum. Our VA does not currently have the ability to mitigate these threshold variances, although the LKF does attempt to adjust for them. The LKF also produces a lag of about 15 iterations that may be due to processing speed. A future improvement is increasing the complexity of the LKF and the VA by making them usable with multiple types of sensors. Because of their simplicity, both the VA and the LKF had to be specifically tailored in the code to the needs of the Kinect, the Servos, and the processing environment. While the simplicity of the theory and algorithms can remain the same, there needs to be added complexity to the code to allow for their use on multiple software and hardware platforms.

The major hurdle that presented the most problems is the lack of accuracy of the Kinect optical/depth sensors. If the sensor is not accurate, then the VA is going to have a limited threshold of accuracy, and if the VA is limited in this way, the LKF can only do so much to transform of the angles so that they more accurately describe the absolute position of the users arm movements. Increasing the quality of the data received, as well as finding alternate sources of joint coordinates are some of the viable solutions to this problem. Other improvements to the movement include adding more degrees of freedom to the shoulder and elbow joints, as well as improving the strength and quality of the servos for increased weights.

Different combinations of the angles between the Kinect skeletons joints can be useful in recognizing specific movements such as sitting down or running. The research here explained is designed to fit the arm systems of the Telebot, a much bigger project that involves an entire humanoid robot. Even so, the design can also be applied to other projects that need similar systems or adapted to the control of completely different structures.

Although the systems used here are not entirely perfect and further development has to be performed to improve its capabilities, the VA and LFK algorithms are a great step to the proper control of humanoid robotic arms.

ACKNOWLEDGMENT

This research is conducted at the Discovery Lab, Florida International University and is supported in part by a donation from Lieutenant Commander Jeremy Robins, who also provided the conceptual theme for this project. We are also thankful to Dyplast Products for their materials donation.

REFERENCES

[1] Guizzo, E., "When My Avatar Went to Work", IEEE Spectrum, vol.47, no.9, pp.26,50, September 2010.

[2] Junichi Sugiyama, Dzmitry Tsetserukou, and Jun Miura, "NAVIgoid: robot navigation with haptic vision", In SIGGRAPH Asia 2011 Emerging Technologies (SA 11). ACM, New York, NY, USA, , Article 9 , 1 pages. 2011

[3] Nakamura, T., "Real-time 3-D object tracking using Kinect sensor", 2011 IEEE International Conference on Robotics and Biomimetics (ROBIO), vol., no., pp.784,788, 7-11 Dec. 2011.

[4] Liying Cheng, Qi Sun, Han Su, Yang Cong and Shuying Zhao, "Design and implementation of human-robot interactive demonstration system based on Kinect", the 24th Chinese Control and Decision Conference (CCDC), vol., no., pp.971,975, 23-25 May 2012.

[5] Van Vuong Nguyen and Joo-Ho Lee, "Full-body imitation of human motions with kinect and heterogeneous kinematic structure of humanoid robot", 2012 IEEE/SICE

International Symposium on System Integration (SII), vol., no., pp.93,98, 16-18 Dec. 2012

[6] Qian, Kun, Jie Niu, and Hong Yang, "Developing a Gesture Based Remote Human-Robot Interaction System Using Kinect", International Journal of Smart Home,Vol. 7, No. 4, July, 2013.

[7] M. Prabakar, and J-H Kim, "TeleBot: Design Concept of Telepresence Robot for Law Enforcement", Proceedings of the 2013 World Congress on Advances in Nano, Biomechanics, Robotics, and Energy Research (ANBRE13), Seoul, Korea, pp. 34-42, August 25-28, 2013

[8] N. Prabakar, C. Tope, and J-H Kim, "A Smart Multi Telepresence Robot Management System", Proceedings of the 2013 World Congress on Advances in Nano, Biomechanics, Robotics, and Energy Research (ANBRE13), Seoul, Korea, pp. 43-51, August 25-28, 2013.