Hand Based Control of a 3-Finger Gripper

Arman Tailakov
Department of Robotics and Mechatronics
SST, NU
Astana, Kazakhstan
arman.tailakov@nu.edu.kz

Ayan Zhansultanov
Department of Robotics and Mechatronics
SST, NU
Astana, Kazakhstan
ayan.zhansultanov@nu.edu.kz

Abstract—This paper is focused on introducing hand based control system using low-cost hardware set up which consist of Arduino Micro microcontroller, multiplexer and 6 6-DOF Polulu Mini IMU sensors. Described mathematical model are designed to be computationally efficient in order to minimize lag from real-time human hand. Paper describes procedure of designing and various characteristics of methods and their respective performances.

Keywords—gripper; Robotiq; glove; IMU; IMU filtering; Blender

I. Introduction

In the modern world, various teleoperation scenario tasks are performed by robots remotely controlled by a human operator. The objective of this project is to examine different gesture recognition solutions and to build a 3-finger Robotiq adaptive gripper controller, using bunch of sensors as custom made sensor glove, depth camera, or leap motion device. Furthermore, findings from precious step will be implemented on the virtual 3-D hand model in the Blender software, and could be sent to the gripper mounted on UR10 robot manipulator to make it grasping exercises. The results of the project can be used as the core for advanced remote control system for the robots which are designed to operate in the conditions highly unsuitable for human labor such as miners, firemen as well as in the cases when high precision performance is required (surgery).

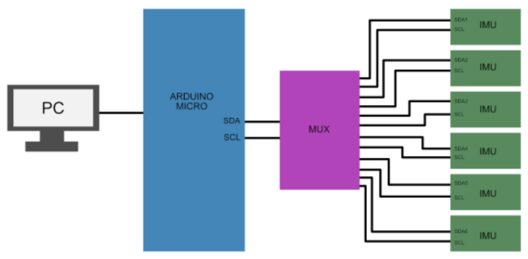


Fig. 1. General schematic of hardware implementation

II. Literature Review

Literature review shows different methods of multi-fingered control techniques of 3-finger gripper, using various sensors increasing efficiency, usability and reliability. Also it shows different methods of filtering and research on constraints of the human fingers.

[1] Peer, Einenkel and Buss (2008) presented and evaluated a point-to-point position mapping and simple force mapping algorithms to control 3-finger robotic gripper using human hand to solve complex and fine manipulation tasks. Moreover, they designed a force feedback using an exoskeleton (CyberGrasp). Also, their dataglove (CyberGlove) uses Flex sensors, which were of interest for our project, and mathematical point-to-point algorithm is provided in the paper.

[2] Kieu, Yamazaki, Hanai, Okada and Inaba (2011) present paper where they try to let human user to teach a robot to do grasping exercises. They use color camera and 3-D sensors for hand tracking task.

[5] Fai Chen Chen et al (2013) presented their research article focused on detailed dynamic and kinematic behavior of the human hand for the mechatronic projects focused on designing hand based systems such as robotic hands. Article consist of ample amount of data and specifics of hand movement in the form of solved direct and kinematic equations for each finger joints which essentially contributes to understanding characteristics of the human hand behavior.

[6] Madgwick et al (2011) presented their paper which introduces orientation filter algorithm developed in order to complement to computationally effective, wearable human motion tracking systems for various applications. Algorithm is suitable for 9-DOF inertial measurement units (IMUs) with 3 axis gyroscope, accelerometer and magnetometer. Based on quaternion representation, algorithm uses raw data from IMU and using gradient descend method estimates the vector of the gyroscope error in the form of quaternion derivative. According to empirical experiments conducted using commercial IMUs Madgwick algorithm provides better performance in comparison with Kalman-based algorithms, and argues that ability to perform in small sampling rates makes the algorithm suitable for designing inexpensive, wearable, lightweight and energy efficient motion tracking systems.

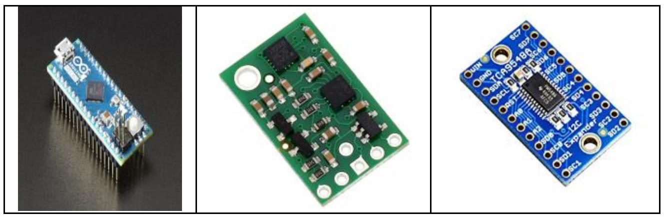
[9] Cobos et al (2007) in their research paper is focused on achieving natural finger and hand motion behavior through real life human hand motion imitation method. In order to collect data about natural constraints various tests using Cyberglove have been conducted to compare with human hand model. Based on the collected data they derived equations which explained existing dependencies of angle joints in the same finger of real human hand. Results of their work can be implemented in various hand based manipulation applications.

III. Sensor Glove

As it was stated in the project proposal, according to the timeline our task in February was to design a custom-made sensor glove, which will be used to read gestures directly from human operator's hand and send acquired information to the controller. First, we needed to decide which hardware equipment we will use. For the microcontroller or simply processing center Aduino Micro 5V, an Open-Source electronic prototyping platform was selected, because it is powerful enough for our task.

To control 3-finger gripper, we need to read position of these 3 fingers relatively to the palm of the human hand. Therefore, at least 4 sensors are needed to be used and consequently wired to the processing center. All these wirings would restrict and limit movements of the hand, and hence we decided to send data from the sensor glove to the Arduino and in future we could use come wireless transmitter to make this product easier to use.

Fig. 2. Arduino Mico, Pololu minIMU v9-5 and TCA9548A I2C Expander

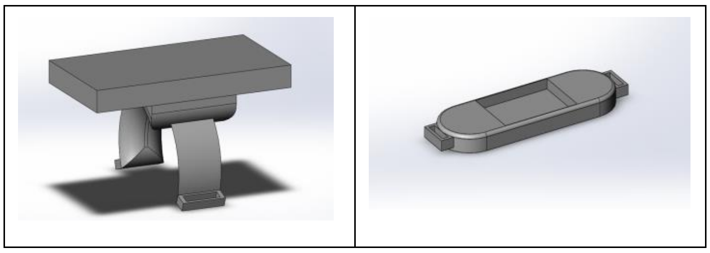


The next issue was to select an appropriate sensor to recognize gestures. First, we sought about using leap motion or 3-d sensors, however their cost is too high, from $20 and $40 respectively for each. The second sought was to use Precision Flex Sensors. Fortunately, we already had couple of gloves with Flex sensors in the laboratory equipment from previous projects, and we started to examine their capabilities. Results were not satisfactory because:

a) Noisy signals
b) Low accuracy
c) Short lifetime

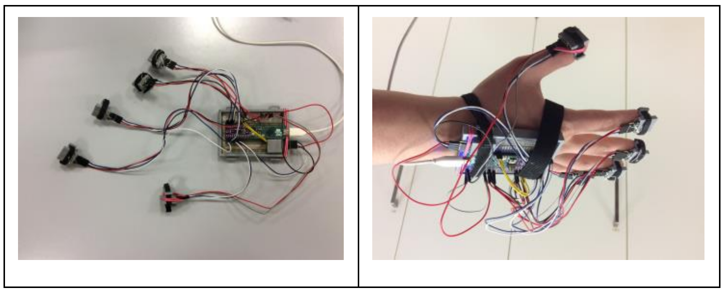
Therefore, we decided not to use Flex sensors, but to use Pololu MinIMU 9 v5 sensors, because they are more reliable, compact and cheaper than other similar motion tracking sensors. However, we still need to connect 3 IMUs for fingers and one for palm of the hand as center of coordinate system, which is a little complicated because there are not enough input pins in the Arduino Mini for all sensors simultaneously. Fortunately, MinIMU sensors use I2C communication protocol, so that we can exploit all needed sensors using only 2 wires. Each sensor in this i2c network will be given a specific address, so that when all sensors will be connected through specific TCA9548A I2C multiplexer, processing center will be able to reach desired sensor by 7-bit address.

Fig. 3. 3-D model of bases for MinIMU and Wixel



The next step was to develop 3-D model of the glove itself. As we already decided to make the glove as compact and comfortable as possible, we designed only small grippers for sensors at ends of each fingers. Also, one IMU will be placed on the palm or on the back side of the hand for considerations of center of the coordinate system, because incorrect orientation of the hand itself can bring errors to the orientation of each finger. Now we need to print the first prototype of the model on 3-D printer and interconnect all parts between each other to start the final steps of this project, namely experimental part.

Fig. 4. Final model of the cyber glove and hardware



IV. 3-D Model of the Hand and Visualization

Before implementing introduced hand based control system on robotic arm, it is essential to verify the realism extent of the kinematic behavior which was achieved by the means of previously mentioned methodologies. For this purpose, widely use 3-D modelling software - Blender is used. Blender incorporates various tools that provides precise 3D modelling capabilities and has built-in Python 3.2 and Game Engine environments for simulation purposes as well. In order to observe motion behavior of the system firstly realistic human hand was designed. It should be noticed that realistic human hand is not only refers to outer appearance of the 3D model, but also refers to the bones inside the 3D model - they need to simulate real human hand bones. The purpose of creating bones is to manipulate the outer appearance of the 3D model. Using built-in Game engine environment commands these bones can be manipulated in various ways. In order to realistically control 3D model Angle joint THETA_DIP is parsed from MATLAB and using infrafinger constraints equations other joints' angles are calculated and linked to respective bones of the 3D model. First results provided that there is a high sensitivity of some IMUs' roll angle, and by the means of mathematical manipulations on these angles the motion of 3D model was configured to exactly match real-time human hand behavior. In order to converge lag of the simulation from real time human hand to zero both MATLAB code and Blender code was written as compact as it can to be and parsing baud rate from Arduino Micro was increased to 9600 bits/second. Baud rate of 9600 bits/second means that system's overall frequency was about 35 HZ, which is satisfactory value to show stable kinematic behavior of the model. Figure 5 shows the final model of the hand, which is required to visualize the final progress of this project

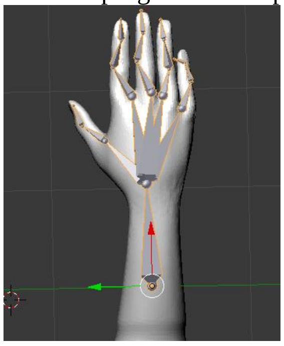


Fig. 5. Final 3-D model of the hand in Blender software

V. Data Calibration and Filtering

The integral part for designing motion tracking systems is to properly calibrate the sensors. We used in this project 6 IMU sensors, and 6 values from each – x, y, z values of accelerometer and gyroscope. Due to manufacturing reasons, all of these values are not clear from the beginning, they have noise, some initial offset and inappropriate range. To calibrate gyro, we firstly collected raw data with 2000 readings when IMUs are at stationary position. It is known that at rest gyro values should approach 0, therefore we subtracted the mean value of all sample values from the raw data. Also, some fluctuations were detected, so we divided raw values by the difference of maximum and minimum of read values. With this we decreased manufacturing noise and offset error from the raw data. For accelerometer, we divided the x, y and z values by the maximum values of the data for each value respectively in order to normalize and simplify output raw data. However, these calibrations are not enough, because there are a lot of different noises resulting in small inevitable fluctuations. Firstly, to solve this problem we planned to implement Kalman filter for our project, which became the accepted basis for majority of algorithms used by scientists and engineers worldwide in the past decades [7]. Then our supervisor advised to take a look on Madgwick filter – a relatively new approach with IMU and AHRS senor fusion algorithm. It was developed by Sebastian Madgwick in 2009, and fuses angular velocities and accelerations from IMU devices into an orientation quaternion. The algorithm is posted in open-source resources in C, C# and MATLAB environments. We decided to use the MATLAB package and implement all computational code there. The translation from the orientation quaternion to the yaw, pitch and roll angles is done using equations (7), (8) and (9) by Madgwick et al in their conference paper [6].

$$\psi = \mathrm{Atan2}\left(2q_2q_3 - 2q_1q_4, 2q_1^2 + 2q_2^2 - 1\right)$$

$$\theta = -\sin^{-1}\left(2q_2q_4 + 2q_1q_3\right)$$

$$\phi = \mathrm{Atan2}\left(2q_3q_4 - 2q_1q_2, 2q_1^2 + 2q_4^2 - 1\right)$$

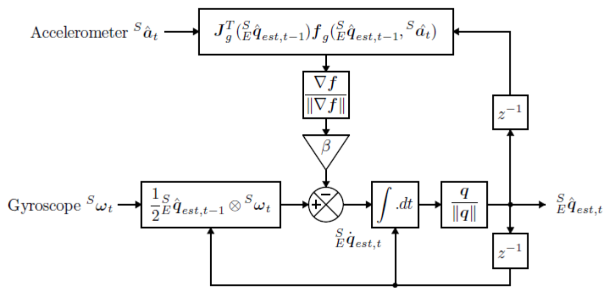
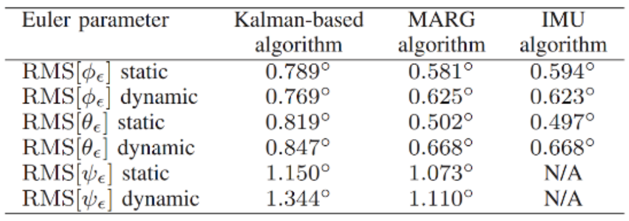

Accelerometer ${}^S\hat{a}_t$
$J_g^T({}^S_E\hat{q}_{est,t-1})f_g({}^S_E\hat{q}_{est,t-1}, {}^S\hat{a}_t)$
$\frac{\nabla f}{\|\nabla f\|}$
β
Gyroscope ${}^S\omega_t$
$\frac{1}{2}{}^S_E\hat{q}_{est,t-1} \otimes {}^S\omega_t$
$\int .dt$
$\frac{q}{\|q\|}$
${}^S_E\hat{q}_{est,t}$
${}^S_E\dot{q}_{est,t}$
z^{-1}
z^{-1}

Fig. 6. Block diagram representation of the complete orientation for an IMU fusion algorithm implementation



Euler parameter	Kalman-based algorithm	MARG algorithm	IMU algorithm
RMS[ϕ_ϵ] static	0.789°	0.581°	0.594°
RMS[ϕ_ϵ] dynamic	0.769°	0.625°	0.623°
RMS[θ_ϵ] static	0.819°	0.502°	0.497°
RMS[θ_ϵ] dynamic	0.847°	0.668°	0.668°
RMS[ψ_ϵ] static	1.150°	1.073°	N/A
RMS[ψ_ϵ] dynamic	1.344°	1.110°	N/A

Fig. 7. Static and dynamic RMS error of Kalman-based algorithm versus MARG and IMU algorithms proposed by Madgwick et al

Figure 4 shows a block diagram of complete implementation of Madgwick filter fusion algorithm developed in [6]. Madgwick et al claim that their algorithm is more precise than widely accepted Kalman filter in number of criteria, such as less sampling rates, smaller computational loads and better precision of output results. Figure 7 illustrates RMS errors of Kalman and Madgwick algorithms in static and dynamic positions, where we can observe that Kalman-based algorithm has 0.1-0.4 degrees larger RMS errors than Madgwick algorithms. Therefore, we decided to use Madgwick filter than Kalman filter.

VI. Deriving Human Finger Kinematics

In order to develop natural kinematic behavior of the robotic hand system without tracking dynamic behavior of each finger joints it is essential to understand specific constraints and characteristics that human hand has. Fai Chen Chen et al [5] in their research article "Constraint Study for Hand Exoskeleton: Human hand dynamics and kinematics" mention "infra finger constraints" term which refers to angular constraints between joint in the same finger. Basically, this term explains that there is a geometrical dependence in the form of simple equations between joint angle in the same finger of the hand. This knowledge leads to the idea that in the case when orientation vectors of the first phalange of the finger and palm are known, it is possible to develop natural kinematic behavior of the finger for simple grasp motion.

This idea has practical usage in the current project, since the IMUs are located exactly at the same places: first phalange of each finger and parallel to the palm. Infrafinger constraint equations used in the project was taken from Cobos et al (2007) paper and are following:

For fingers: $\theta_{\mathrm{DIP}} \approx \frac{2}{3} * \theta_{\mathrm{PIP}} * \theta_{\mathrm{PIP}} \approx \frac{3}{4} * \theta_{\mathrm{MCP}f/e}$ (1)

Where DIP stands for distalinterphalangeal joint, PIP stands for proximal interphalangeal joint, MCP stands for metacarpophalangeal joint.

For thumb: $\theta_{\mathrm{IP}} \approx \frac{1}{2} * \theta_{\mathrm{MCP}f/e}$, $\theta_{\mathrm{MCP}f/e} \approx 5/4 * \theta_{\mathrm{TMC}f/e}$ (2)

Where IP stands for interphalangeal joint, MCP stands for metacarpophalangeal joint, TMC stands for trapeziometacarpal joint.

Even though Fai Chen Chen et al [5] argue that infrafinger constraints introduced in Cobos et al [9] work may not appear to be strict enough, it still provides with satisfactorily accurate equations in case of simple grasp motion which is implemented in the project.

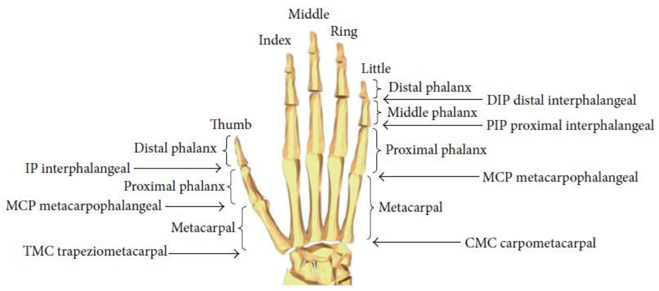


Fig. 8. Anatomical Details of the Human Hand

VII. Conclusions

According to the results of conducted research on hand based control of robotic arm it is suggested that this device has the capability to be used in various applications which are not necessarily have to be robotic hand, but any robotic system that is in need for precise and intuitive control by the human hand. Current set up aimed to control three fingered robotic hand, and thus has mathematical model which is limited to compute simple grasping motion of the hand. The results of simulation of grasping motion are satisfactorily precise and fluid. In order to design more flexible device, it is suggested to modify existing mathematical model which would take into account multiple angles of rotations and advance infrafinger constraints. Addition of wireless modules as WiFi or Bluetooth can potentially enlarge customer spheres in case of commercial usage.

VIII. Acknowledgements

We would like to thank Alikhan Zhilisbayev and Akan Takish for sharing a complete 3-D model of Human Hand in Blender.

IX. References

[1] A. Peer, S. Einenkel and M. Buss, "Multi-fingered telemanipulation - mapping of a human hand to a three finger gripper," *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, Munich, 2008, pp. 465-470.

[2] T. C. Kieu, K. Yamazaki, R. Hanai, K. Okada and M. Inaba, "Grasp observation and reproduction by humanoid robots using color camera and 3D sensor," *2011 IEEE/SICE International Symposium on System Integration (SII)*, Kyoto, 2011, pp. 808-813.

[3] J. M. Romano, K. Hsiao, G. Niemeyer, S. Chitta and K. J. Kuchenbecker, "Human-Inspired Robotic Grasp Control With Tactile Sensing," in *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1067-1079, Dec. 2011.

[4] J. K. Sharma, R. Gupta and V. K. Pathak, "Numeral Gesture Recognition Using Leap Motion Sensor," *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, Jabalpur, 2015, pp. 411-414.

[5] Fai Chen Chen, Silvia Appendino, Alessandro Battezzato, Alain Favetto, Mehdi Mousavi, and Francesco Pescarmona, “Constraint Study for a Hand Exoskeleton: Human Hand Kinematics and Dynamics,” Journal of Robotics, vol. 2013, Article ID 910961, 17 pages, 2013. doi:10.1155/2013/910961

[6] Sebastian O. H. Madgwick; Andrew J. L. Harrison; Ravi Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm", 2011 IEEE International Conference on Rehabilitation Robotics, pp. 1 - 7, DOI: 10.1109/ICORR.2011.5975346

[7] R. E. Kalman. A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82:35–45, 1960.

[8] S. A. Dalley, H. A. Varol and M. Goldfarb, "A Method for the Control of Multigrasp Myoelectric Prosthetic Hands," in *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 20, no. 1, pp. 58-67, Jan. 2012.

[9] S. Cobos, M. Ferre, M. Sanchez, and J. Ortego, “Constraints for realistic hand manipulation,” in Proceedings of the 10th Annual International Workshop on Presence (PRESENCE ’07), pp. 369– 370, Barcelona, Spain, October 2007.

X. Appendices

APPENDIX 1. MATLAB CODE

```
addpath('quaternion_library');
clear; clc; close all;

% Copyright 2014 The MathWorks, Inc.

%% Create serial object for Arduino
s = serial('COM7'); % change the COM Port number as needed

%% Connect the serial port to Arduino
s.InputBufferSize = 1024; % read only one byte every time
try
    fopen(s);
catch err
    fclose(instrfind);
    error('Make sure you select the correct COM Port where the Arduino is connected.');
end

results=zeros(6,6);
% madgwick filter
    AHRS1 = MadgwickAHRS('SamplePeriod', 1/256, 'Beta', 0.1);
    AHRS2 = MadgwickAHRS('SamplePeriod', 1/256, 'Beta', 0.1);
    AHRS3 = MadgwickAHRS('SamplePeriod', 1/256, 'Beta', 0.1);
    AHRS4 = MadgwickAHRS('SamplePeriod', 1/256, 'Beta', 0.1);
    AHRS5 = MadgwickAHRS('SamplePeriod', 1/256, 'Beta', 0.1);
    AHRS6 = MadgwickAHRS('SamplePeriod', 1/256, 'Beta', 0.1);

    q = zeros(6,4);
    euler = zeros(6,3);
    axang = zeros(6,4);
    ang = zeros(6,3);
col =[4107 4107 4107 120 135 130
      4175 4175 4175 256 256 137
      4200 4200 4200 114 255 270
      4352 4352 4352 164 179 157
      4100 4100 4100 113 184 256
      4150 4150 4150 95 120 85];

offset = 1000*[    -0.9608   -2.0833   -3.3222    0.0769   -0.1119   -0.0904
   -1.7273   -2.9752    2.2720   -0.0492   -0.0289   -0.0726
   -0.2911    0.0576    4.2002    0.0435   -0.0327   -0.2093
    0.4403    0.0495    4.0917    0.1181   -0.1146   -0.1040
    0.5702    4.0171    0.7467    0.0275   -0.0654    0.0529
   -0.0400   -0.2685   -4.0778    0.0271   -0.0907    0.0136];

diff =[    82   510   395    97   119    73
   118   105   337   248   254   115
   328    99   166   130   252   448
   121   319   356    81   143    82
    98   113   505    79   215   103
   240    98   510   242   173   256];

  t = tcpip('127.0.0.1', 5005, 'NetworkRole', 'client');

% num=0;
g=zeros(6,6);
```

```
th1=zeros(5,1);
% max=zeros(6,6);
% min=ones(6,6)*10000;
nn=ones(6,3)*180;
fopen(t);
while 1
    %tic

    %% Read buffer data

for i = 1:6
%     tic
    data = fgetl(s);
    temp = str2double(strsplit(data));
    while length(temp)<8
        temp = str2double(strsplit(data));
    end
    results(temp(1)+1,:) =temp(2:7);
%     disp(toc)
end

% for i = 1:6
%     for j = 1:6
%         if max(i,j)<abs(results(i,j));
%            max(i,j)=abs(results(i,j));
%         end
%         if min(i,j)>abs(results(i,j));
%            min(i,j)=abs(results(i,j));
%         end
%     end
% end

results;
% if num<2000;
%    g=g+results;
% end
results(:,4:6) = results(:,4:6)-offset(:,4:6);
results(:,4:6) = results(:,4:6)./diff(:,4:6);
results(:,1:3) = results(:,1:3)./col(:,1:3);

        AHRS6.UpdateIMU([results(6,4) results(6,5) results(6,6)] , [results(6,1) results(6,2) results(6,3)]);
        q(6,:) = AHRS6.Quaternion;

        AHRS1.UpdateIMU([results(1,4) results(1,5) results(1,6)] , [results(1,1) results(1,2) results(1,3)]);
        q(1,:) = AHRS1.Quaternion;

        AHRS2.UpdateIMU([results(2,4) results(2,5) results(2,6)] , [results(2,1) results(2,2) results(2,3)]);
        q(2,:) = AHRS2.Quaternion;

        AHRS3.UpdateIMU([results(3,4) results(3,5) results(3,6)] , [results(3,1) results(3,2) results(3,3)]);
        q(3,:) = AHRS3.Quaternion;

        AHRS4.UpdateIMU([results(4,4) results(4,5) results(4,6)] , [results(4,1) results(4,2) results(4,3)]);
        q(4,:) = AHRS4.Quaternion;

        AHRS5.UpdateIMU([results(5,4) results(5,5) results(5,6)] , [results(5,1) results(5,2) results(5,3)]);
        q(5,:) = AHRS5.Quaternion;
    for n =1:6
```

```
        euler(n,:) = quatern2euler(quaternConj(q(n,:)) * 180/pi);
        axang(n,:) = quat2axang(q(n,:));
        axang(n,4) = axang(n,4)*180/pi;
        [y, p, r] = quat2angle(q(n,:));
        ang(n,:) = [y p r];
    end
    ang = ang*180/pi;
    %ang = ang+nn;
    th1 = ang(1:5,2)-[1 1 1 1 1]'*ang(6,2);
    q;
    ang;
    th1';
    results;
    axang;
    euler(1,:);
    results(1,4:6);
    q(1,:);
    q(2,:);
    axang(1,:);
    ang(1,:);

    tht = (81*[1 1 1 1 1]'+4*14*9*abs(th1)).^0.5;
    thp = (tht-9*[1 1 1 1 1]')/28;
    thd = thp.^2*2/3;
    thd(5)=8/23*th1(5);
    a=mat2str(floor(thd'));
    fwrite(t,a);

     % use conjugate for sensor frame relative to Earth and convert to degrees.

end
%% close files
fclose(t);
fclose(s);
delete(s);
clear s;
```

APPENDIX 2. PYHTON CODE IN BLENDER

```
import bge
import socket
import math
from time import sleep
TCP_IP = '127.0.0.1';
TCP_PORT = 5005;
BUFFER_SIZE = 4000; # Normally 1024, but we want fast response
bge.logic.setMaxLogicFrame(1)


class Server:
    def __init__(self):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM);
    s.bind((TCP_IP, TCP_PORT));
    s.listen(1);
    state=0;
    self.conn, addr = s.accept()
    print('Connection address:', addr);


    def receive(self):
        data = self.conn.recv(BUFFER_SIZE);
        return data


server=Server();

def main():
    scene = bge.logic.getCurrentScene();
    source = scene.objects;
    main_arm = source.get('Armature');
    main_arm.update();
    thp=[0,0,0,0,0]
    thm= [0,0,0,0,0]
    thd= [0,0,0,0,0]
    data=server.receive();
    #if not data: data=0;
    if not data: bge.logic.endGame();
    state_value=0;
    main_arm.update();
    str = ''
    if len(data)>=8:
        for dat in data:
            if dat==91 or dat==93:
                dat=32
            str+=chr(dat)
        thd = [int(s) for s in str.split() if s.isdigit()]
    print(thd)
    #5th finger

    #theta_dip_0 = roll angle (around y axis)
    thp[0] = math.sqrt(thd[0]*3/2)
    thm[0]= 4/3*thd[0]
    main_arm.channels['Bone.021'].joint_rotation=[ 0, 0, -thd[0]*math.pi/180];
    main_arm.channels['Bone.017'].joint_rotation=[ 0, 0, thp[0]*math.pi/180];
    main_arm.channels['Bone.016'].joint_rotation=[ 0, 0, thm[0]*math.pi/180];

    #4th finger

    thp[1] = math.sqrt(thd[1]*3/2)
    thm[1]= 4/3*thd[1]
    main_arm.channels['Bone.015'].joint_rotation=[ 0, 0, thd[1]*math.pi/180];
    main_arm.channels['Bone.014'].joint_rotation=[ 0, 0, thp[1]*math.pi/180];
    main_arm.channels['Bone.013'].joint_rotation=[ 0, 0, thm[1]*math.pi/180];

    #3rd finger

    thp[2] = math.sqrt(thd[2]*3/2)
    thm[2]= 4/3*thd[2]
    main_arm.channels['Bone.012'].joint_rotation=[ thd[2]*math.pi/180, 0, 0];
    main_arm.channels['Bone.011'].joint_rotation=[ thp[2]*math.pi/180, 0, 0];
    main_arm.channels['Bone.010'].joint_rotation=[ thm[2]*math.pi/180, 0, 0];

    #2nd fingers)
    thp[3] = math.sqrt(thd[3]*3/2)
    thm[3]= 4/3*thd[3]
    main_arm.channels['Bone.009'].joint_rotation=[ 0, 0, -thd[3]*math.pi/180];
main_arm.channels['Bone.008'].joint_rotation=[ 0, 0, -thp[3]*math.pi/180];
main_arm.channels['Bone.007'].joint_rotation=[ 0, 0, -thm[3]*math.pi/180];
    #1nd finger (THUMB)
thm = 5/4*thd[4]/2;
thi= 5/8*thd[4]/2;
main_arm.channels['Bone.019'].joint_rotation=[ thi*math.pi/180, 0, 0];
main_arm.channels['Bone.018'].joint_rotation=[ thm*math.pi/180, 0, 0];
main_arm.channels['Bone.006'].joint_rotation=[ thd[4]*math.pi/180, 0, 0];
```