CLUSTER INNOVATION CENTRE, DELHI UNIVERSITY

SUMMER INTERNSHIP Project Report on

GESTURE CONTROLLED ROBOTIC ARM

Upasana Singh [11633] | Vaibhav Jain [11634]

III.7 Summer Internship: Projects Drawn from the World Around Us
B.Tech - Information Technology and Mathematical Innovations
[2016-2020]

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our mentor Prof. Shobha Bagai who gave us the golden opportunity to work on this project, which also introduces us to many new terms of robotics. We would also like to thank all the faculty members, who have always been a source of motivation. This project required huge amount of work, and dedication. The project still needs some more work to be done to make it more efficient which could not be completed in the given course of time, but we will try to work on them further. There are many other people also involved in this project therefore we would like to extend our sincere gratitude to all of them.

CONTENT

1. ABSTRACT

Controlling the trajectory of a manual non-industrial robotic arm is a difficult task since its motion changes dynamically. This project aims at developing a user-friendly interface to control a robotic arm using the technique of gesture recognition. In the end, a user has to just stand in front of the camera and move his hands. The robotic arm will try to mimic the motion of the user's hand. It allows fluid movements of arm possible which are not easy to perform using even high-end controller.

A Microsoft Kinect sensor is used for collecting skeletal data of user arm which is then processed and converted into a set of values of angles for each individual joint in robotic arm. The angles of each joint in the arm are controlled by the Arduino's Spider controller. The algorithm used to generate angles tries to achieve the same relative position in space as the user's arm. This makes the user experience very simple, even for a first time user.

2. INTRODUCTION

Nowadays, Robotics have become one of the most promising and fast growing branch in the field of technology. The applications of Robotics include almost every modern technology such as automobiles, medical technology, construction, defence, space probes, fire-fighting robots etc. One of the major class of robots is Robotic Arm. A Robotic arm is a mechanical arm programmed to perform particular tasks similar to human arm with more precision and strength. Robotic arms are typically classified into two types on the basis of their application namely, Industrial and nonindustrial. Industrial arms are generally designed to work in factories/industries where they have to repeat same motion again and again repeatedly. Therefore programming their motion is easy. On contrary, Nonindustrial arms have no predefined motion. It changes dynamically. These types of arms can be mounted on top of some other vehicle, in medical surgery and in many other places. In some situations it is possible to mathematically define the desired trajectory, but this process requires a lot of computation and thus, a great deal of time. More often such delay in motion is not acceptable. To address this problem, research has been going on in past few years on gesture controlled robotic arm.

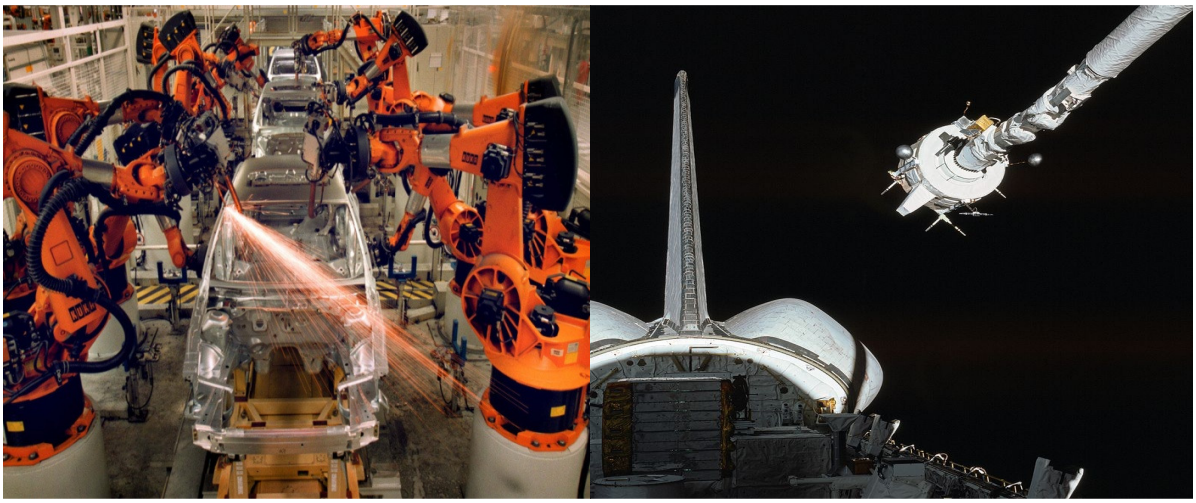

Figure-1: Example of an Industrial Robotic arm working on a conveyor belt in a factory and a non-industrial Robotic arm mounted on a space probe.

The idea of controlling robot through gestures saves a lot of calculations about kinematics and trajectory planning of robotic arm. In addition, it allows arm incapable of following more complicated and sophisticated trajectory without much computations which otherwise might took time due to its heavy dependency on matrix calculation. It also increases the arm's response time to any obstacle in its environment.

In this project we tried to implement this technique of controlling a robotic arm through human gesture on a prototype robotic arm. We used 'Dexter ER-2 Heavy Duty Robotic Arm' by Nex-Robotics as our model arm and 'Kinect' by Microsoft for collecting 3-Dimensional data about user's arm. Objective of this project was to develop a user-friendly interface to control a robotic arm using the technique of gesture recognition. In the end, a user has to just stand in front of the camera and move his hands. The robotic arm will try to mimic the motion of the user's hand.

3. PREVIOUS WORKS

Before explaining our work, we would like to throw light on some projects that work on the same idea of controlling a Robotic arm through Human gestures, and the way they did it.
The projects are:

3.1 E-Yantra Gesture controlled Robot

This project was taken in E-Yantra 2012. In this project, they have used a Fire-Bird module with gripper extension to perform tasks along with Kinect to capture gestures. The project used Open CV for the motion tracking. Basically the firebird and gripper are controlled using gestures given by the user. Flow of control of this project can be summarized as follows:

- The Kinect captures the gestures.
- The laptop sends appropriate signals to firebird over Zigbee Bluetooth module.
- The firebird now processes these signals and takes appropriate actions

Figure-2: An operator demonstrating the working of Gesture Controlled Robot

3.2 MEMS based Gesture Controlled Robot Arm

A Micro Electro Mechanical System is an industrial technology for small devices. This project reports an adoption of this technique for transmitting gestures to a military robot to control its functions. The robot in this project comprises of three parts, MEMS sensor, AT89S52 microcontroller and motor driver. This gesture controlled robotic arm project is self controlled and the sensor is fixed to the arm that includes an accelerometer.

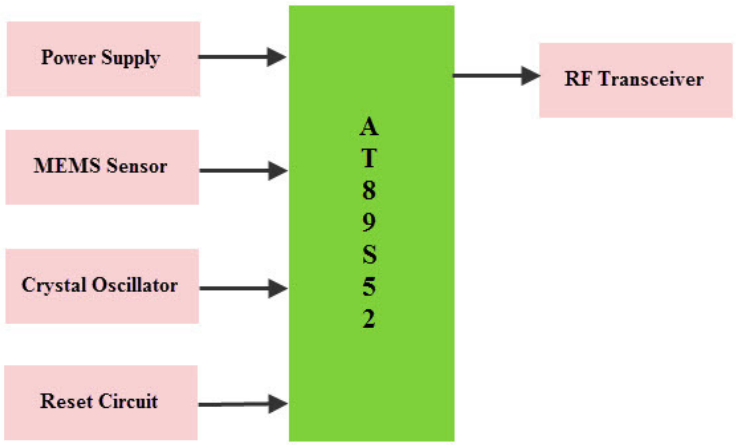

Figure-3: MEMS based Gesture Controlled Robotics Arm Transmitter

This project includes: transmitter section and receiver section. At the transmitter end, the microcontroller receives signals and sends it to the receiver end through RF transceiver. At the receiver end, the microcontroller receives the signals from the RF transceiver and the motor is controlled by the motor driver.

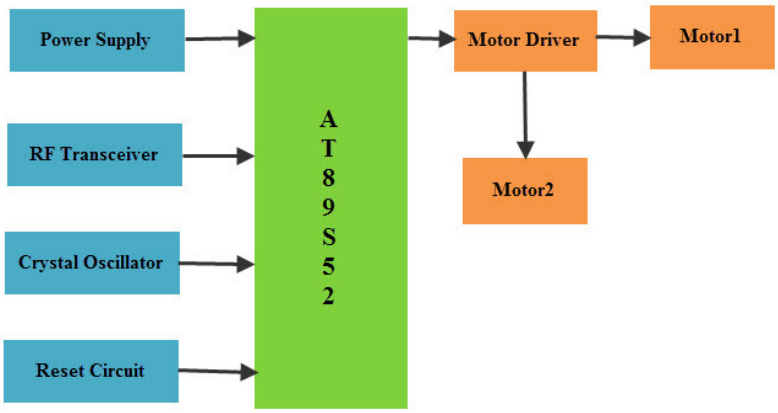

Figure-4: MEMS based Gesture Controlled Robot Arm Receiver

3.3 Hand Gestures Remote Controlled Robotic Arm (Bharati Vidyapeeth College of Engineering, New Delhi)

In this project the Robotic Arm is controlled wirelessly with hand gestures with the help of a specially designed glove.
The components used are:

- The Robotic arm has 6 servos and 3DOF's (Degrees of Freedom)
- Arduino UNO for taking values from the module and sending data to servos

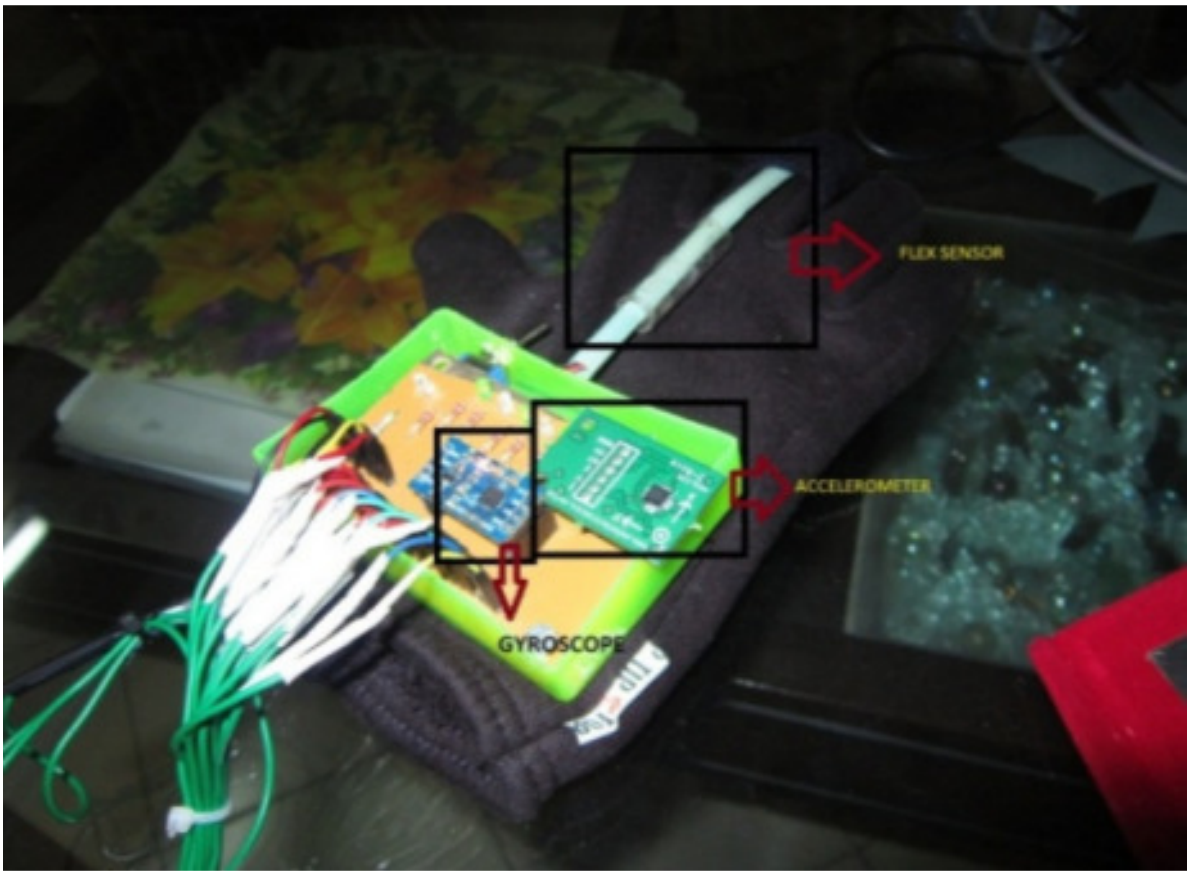

Figure-5: The figure shows flex sensor, accelerometer & Gyroscope as parts of the transmitter fixed on the hand glove

This project simply changes the mode of data transferred. In this the glove is worn by a user who moves his hand. The glove is designed in such a way that it has internal circuits which receive the data and transmit it to the arm with the help of Arduino UNO.

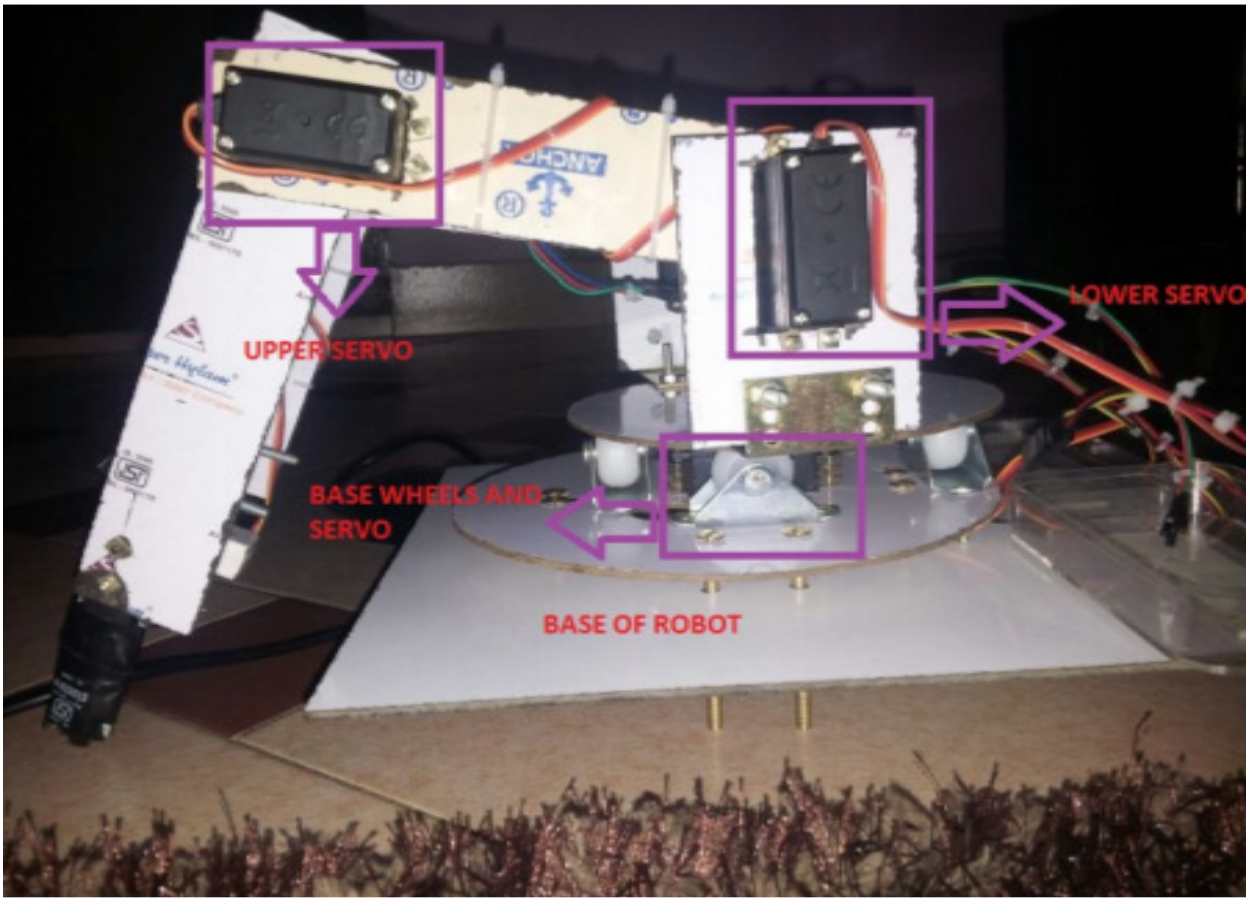

Figure-6: Complete assembly of the Robotic Arm used in this project

4. RESOURCES USED

4.1 HARDWARE

KINECT by Microsoft

Kinect is an SDK (Software Development Kit) for Windows. It is a motion detecting camera and it uses special technology to track body movements and translate those movements to the arm. Besides gesture control, Kinect is mainly used for games and sometimes for voice recognition. We are using Kinect to take the arm movements of the user as inputs and pass on the data to the microcontroller to transfer the data to the arm.

Figure -7: Microsoft's Kinect Sensor

HOW DOES IT WORK? According to Microsoft, Kinect is a hands-free motion control device. The Kinect sensor is a horizontal rod connected to a motorized bottom that is placed somewhere near the monitor/television set. Built into the rod is a **multi-array microphone**, an **RGB camera** and a **depth sensor**. These three elements together help the Kinect to perform 3D motion capture and facial and voice recognition.

The depth sensor consists of an infrared projector and sensor. The projector projects a continuous infrared pattern over its field of vision which the sensor uses to interpret the view. Facial, motion and voice recognition are handled by Kinect Software. For our project we are only using motion recognition of Kinect. The RGB camera records the hand movement of the user and passes the data to the attached microcontroller for the arm to follow the pattern.

Dexter ER-2 Heavy Duty Robotic Arm

Dexter ER-2 Robotic Arm is a 5 Axis robotic Arm designed by NEX Robotics. It has 4 metal gear servo motors with 15Kg/cm torque and two servo motors with 7Kg/cm torque. It is of 5 DOF (Degree of Freedom), which includes: Base rotation, Shoulder rotation, Elbow rotation, Wrist pitch and the gripper.

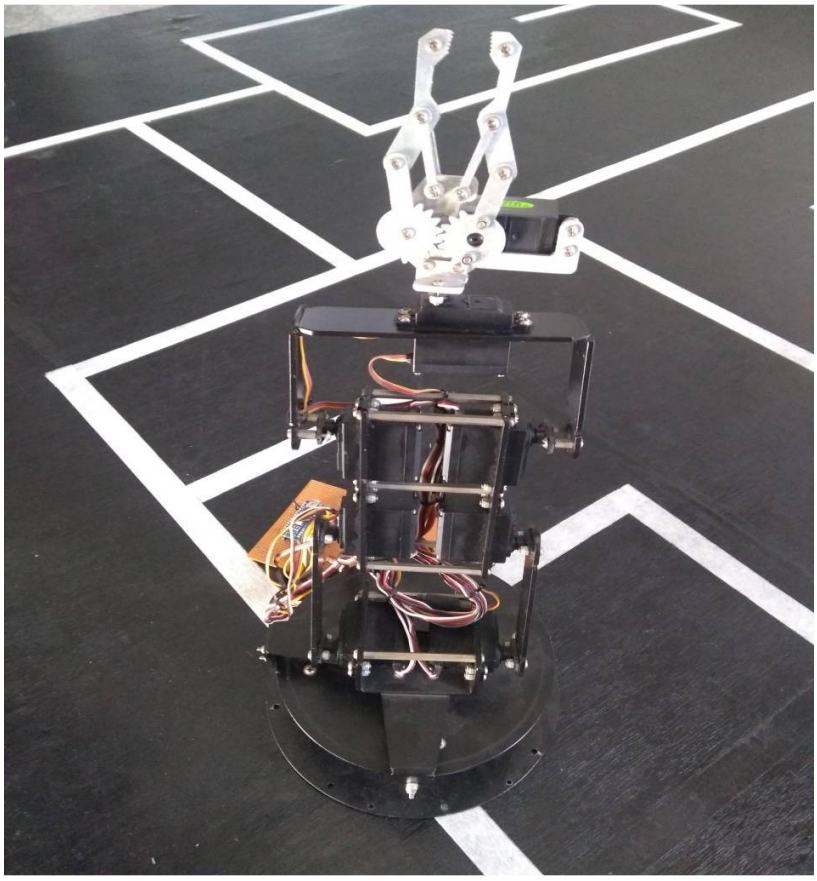

Figure - 8: Dexter ER-2 Heavy Duty Robotic Arm

Robotic Arm comes with 32 channel universal servo motor controller board and GUI. Servo control board can control 32 servo motors simultaneously. But here instead of using the GUI and the universal microcontroller we are using a self made PCB consisting of Arduino's NANO microcontroller board. The arm can be programmed to execute different types of motion profiles using any of the 5 servo channels simultaneously.

Printed Circuit Board (PCB)

We built a Printed-Circuit-Board (PCB) which is inspired by the design of Spider Controller. We have specially designed it for the Dexter Arm. It can connect up to 9 servos. It has three headers just as in Spider Controller for Signal, Vcc and Ground pins respectively (starting from inner side). We are using Arduino Nano as microcontroller with this circuit.

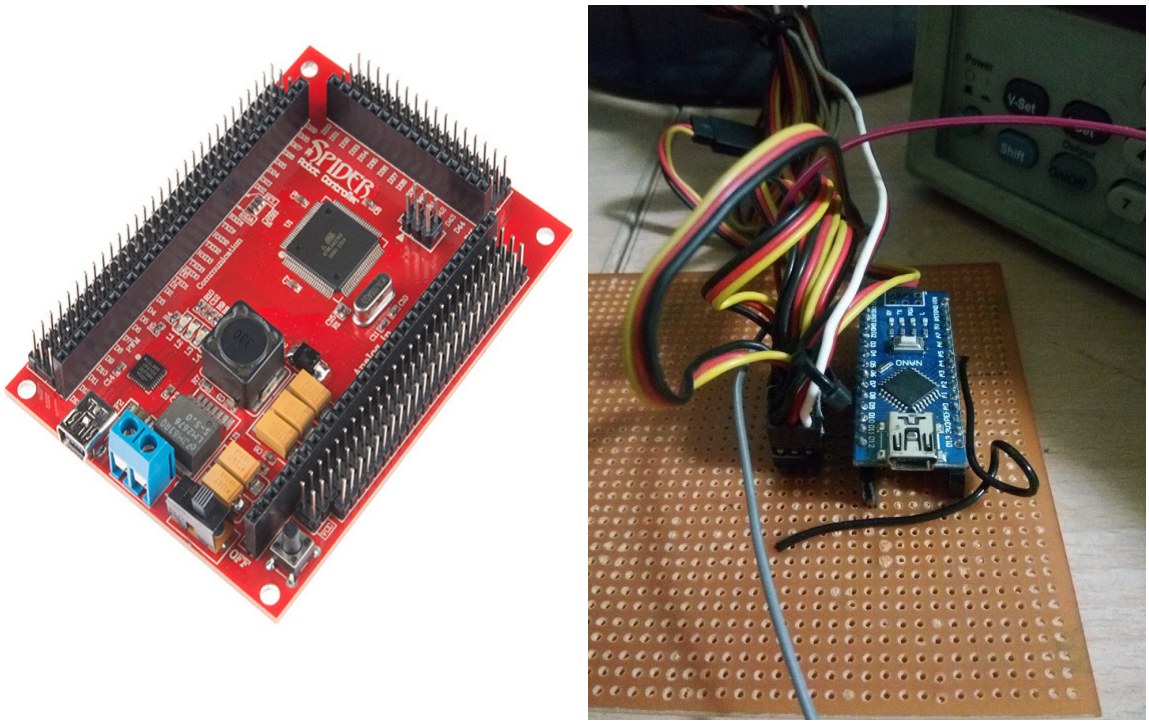

Figure - 9: a) Arduino's Spider Controller and b) indigenously built PCB

4.2 SOFTWARE:

ROBOT OPERATING SYSTEM (ROS)

ROS is an open-source, meta-operating system for robots. It provides services like hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management, to the operating system of your device. Moreover, it also provides tools and libraries for building, writing, and running code across multiple computers. ROS provides libraries and tools to help software developers create robot applications.

We have used ROS-kinetic version of ROS. It is among the latest and currently the most recommended distribution of ROS.

ARDUINO ENVIRONMENT (Arduino-IDE)

The Arduino Integrated Development Environment is basically a text editor for writing code. It connects to the Arduino and Genuine hardware to upload programs and communicate with them. Programs written using Arduino Software (IDE) are called sketches. These are written in an Arduino text editor and are saved with '.ino' extension. Then these files are uploaded to the Arduino board through IDE.

5. METHODOLOGY

First matter we need to address is to analyze the structure of a human arm and its representation through DEXTER arm. In a human arm, the shoulder has a ball and socket joint having 2 degrees of freedom which mechanically, can be represented by two separate revolute joints connected in series with each other. Therefore the Servo 1 and Servo 2 of DEXTER arm combined represent the shoulder of the human arm. The elbow of a human arm of a simple revolute joint has 1 degree of freedom which we represent by Servo 3 of arm. For the sake of simplicity, we ignore all further servos on the arm to maintain their orientation as it is. Now we are ready to start collecting data from Kinect.

5.1 MOTION TRACKING OF USER'S ARM USING OPENNI TRACKER IN ROS

ROS software is organized into packages each of which contains some combination of code, data and documentation. ROS ecosystem consists of thousands of publicly available packages in open repositories. One of them is an OPENNI_TRACKER package by Tim Field. This package tracks human skeleton in front of the Kinect and broadcasts the data in tf format. The tf:: quaternion format is a data structure which is used to keep data of multiple coordinated frames over time. A robotic system in ROS typically has many 3D coordinates' frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. tf keeps track of all these frames over time.
OPENNI_TRACKER allows us to get transformed matrices between frames of camera and user's arm joints i.e. shoulder, elbow along with other information like timestamp on data, orientation of axis etc. Following figure shows an image of a human skeleton tracked by OPENNI_TRACKER.

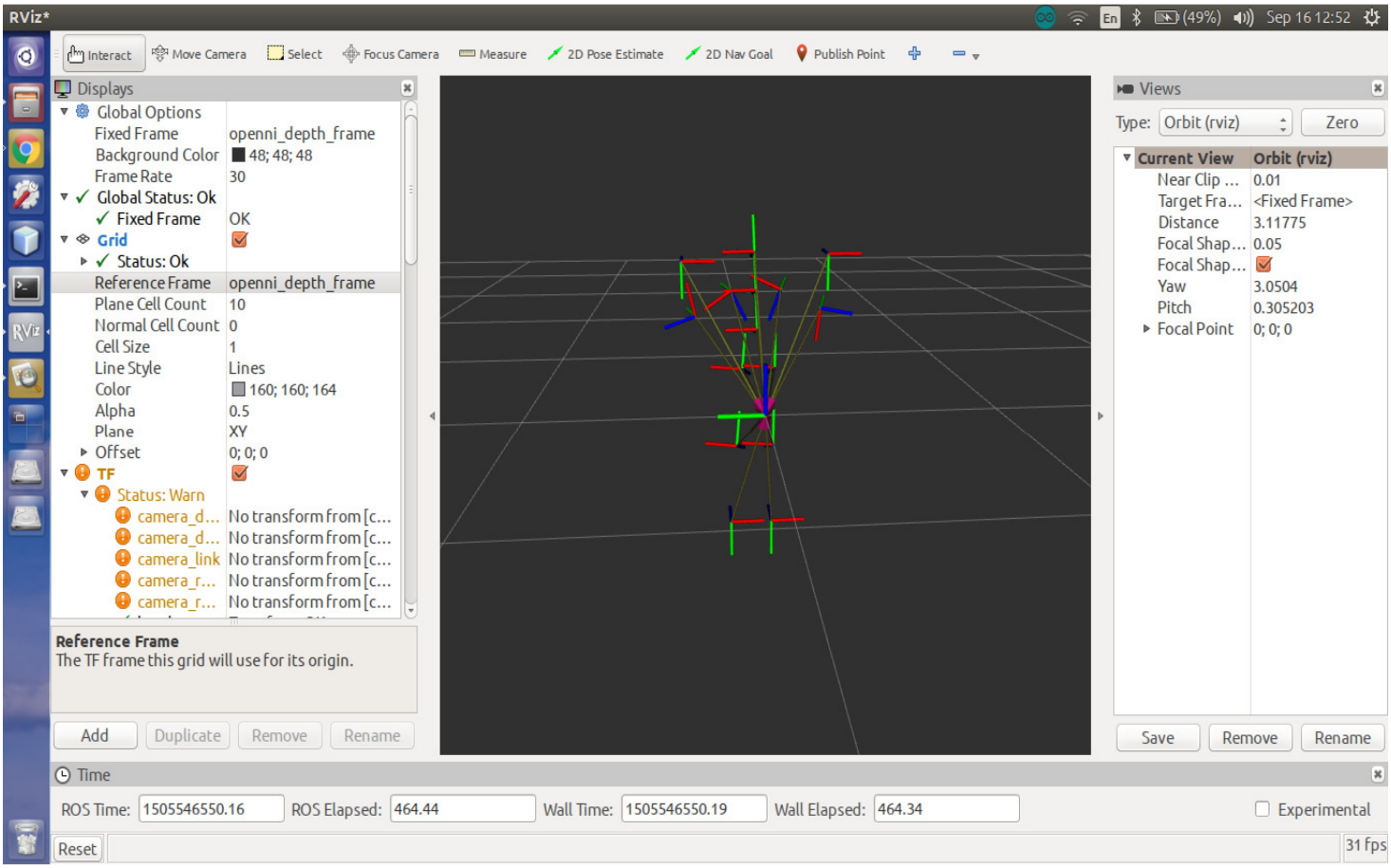

Figure - 10: Representation of Human skeletal obtained through Kinect

Essentially, we are only concerned about these following transformations since our model deals with only arm's section of a human user:

a) Between camera and user right shoulder
b) User right shoulder and user right elbow
c) User right elbow and user right wrist

5.2 CONVERTING DATA TO Roll-Pitch-Yaw (RPY)

The data obtained from OPENNI_TRACKER in tf::quaternion form is now to be converted into some other notation to have some useful information for manipulating the arm. To do this we first convert tf::quaternion into another class of tf package called tf::matrix. This transition between different classes of tf allows us to get ROLL, PITCH and YAW of the transformation between two frames obtained from OPENNI_TRACKER.

The tf::matrix class supports this functionality through a function in its library which converts a rotation matrix into Roll, Pitch, Yaw. This function “getRPY()” takes three variables of the double data type and feeds the value of RPY into them.

5.3 SENDING DATA TO ARDUINO THROUGH SERIAL PORT

Since the arm is controlled by Arduino, we now need to transmit the RPY obtained between each frame to Arduino. The communication between ROS and Arduino is done through the serial communication which provides ROS a communication protocol which allows it to be able to work with Arduino’s serial ports.
UNIX based systems, when an Arduino Board connects through a serial port, it creates a file ‘/dev/ttyACM0’ on the system. ‘/dev/ttyACM0’ is a USB communication device (CDC) of subtype "abstract control model" (ACM). This file contains the data which we send to Arduino over serial port. USB CDC (Communications Device Class) ACM (Abstract Control Model) is a publicly documented protocol that can be used for mirror serial ports over USB. We write the RPYs of each frame which we want to send to Arduino on this file.
Over in the Arduino, we receive this data through the serial monitor which is an Arduino environment's built-in monitor to communicate with Arduino Board. Arduino IDE contains functions like ‘Serial. reads ()’ which reads the value from serial monitor and copies it into its parameter variables.

5.4 CONVERTING PROJECTIONS OF Roll-Pitch-Yaw (RPY) TO EULER ANGLES

At this point, we have RPYs of each of the transforms required to move the arm. But the servo motors are controlled by providing exact angles in degree we want it to be. Here comes the challenge of converting RPY of shoulder and elbow frames into angles for Servo 1-3 of DEXTER arm such the arm’s orientation is same as that of the user’s arm.

To do this, we consider the shoulder of human arm as a 3 - dimensional vector. We define the vertical as its z-axis. Therefore, the angle between x-axis and its projection on x-y plane is the orientation angle of the Servo 1 motor. Mathematically, if we denote the shoulder vector by v then:

$$V_x = \cos(\text{yaw}_{\text{shoulder}}) \times \cos(\text{pitch}_{\text{shoulder}})$$

$$V_y = \sin(\text{yaw}_{\text{shoulder}}) \times \cos(\text{pitch}_{\text{shoulder}})$$

$$V_z = \sin(\text{pitch}_{\text{shoulder}})$$

therefore angle θ between the projection of v and x-axis is given by:

$$\theta 1 = \text{atan}\ (\ V_y / V_x\ )$$

For Servo 2, the angle required is the one made between the vector and z-axis. This angle is same as the Roll angle.

$$\theta 2 = (\text{Roll})_{\text{Shoulder}}$$

Similarly, Servo 3 also has the same angle as that of Roll angle of its transform i.e.

$$\theta 3 = (\text{Roll})_{\text{Elbow}}$$

5.5 WRITING ANGLES TO SERVOS

Now that we have the values of joint angles for each servo, what is left is to write these angles to the servo motors. Angle to the servo is given by creating an object of Servo type which have an inbuilt function called Servo::write(). This function takes an angle in degrees as its parameter and rotates the servo by that angle from its starting position.

6. FUTURE WORK

Further to add to our project, we would like to work more on its accuracy.
Since the movements of the arm are yet not smooth and accurate, we would try to improve its algorithm and make it more efficient. Moreover, since the arm requires a manual start, we can try to start it automatically. What we can do is storing a specified gesture (like keeping your left hand at an angle of 90 degree from your front face) as the starting position. Now once all the connections are made and the Kinect detects this stored gesture, it will start imitating, i.e., it will start functioning. Before this the camera will just be switched on, the stored position of arms is to calibrate the Kinect and inform it to start following the hand gestures of the user.
One can also work on reducing the size and weight of the arm to make it easy to carry while deploying on another vehicle or mobile robot.
Also, sometimes while tracking the human skeleton the camera loses its focus and fails to retrieve user motion. This can also be improved by improving the motion tracking algorithm of the arm.

7. CONCLUSION

Technology advances with time. The things which were considered almost impossible in the past now have become the truth of the present. Recently, autonomous robotic especially Gesture controlled robots have emerged as a hot field of study and research. In the present scenario robotic arms are used only as research environment.

In this project, we were able to program a prototype robotic arm model to be able to operate through gestures of a human arm. But still there is a lot of scope in advancement and progress in this field. Through this project, we were able to learn about the kinematics and trajectory planning of robotic manipulators. This project also introduced us to many new existing technologies like Arduino and Robot Operating System (ROS) which will help us for many other projects in future.

Although, this project still have not addressed many problems which restricts it to be used in the real world. One of those problems is robust motion tracking algorithm. Many times the camera lose human tracking due to environment noise which is not acceptable in real time use. Another important factor to consider is the smoothness of motion of robotic arm.

Despite these challenges, we will continue to progress in this project. Work on constructing better motion tracking algorithm has already started and we expect to report results rather soon.

8. APPENDICES

APPENDIX A: Arduino Sketch

```
#include <Servo.h>
#include <stdlib.h>
#include <math.h>

#define pi 3.141

using namespace std;

Servo servo[9];

//=====================================//
// Initial Angles (Of all six joints for DEXTER ROBOTIC ARM by NEX ROBOTICS) .....90...90...80...80...80...90
//=====================================//
int a1 = 90; //gripper
int a2 = 90; //wrist
int a3 = 80; //elbow
int a4 = 80; //elbow
int a5 = 80; //shoulder
int a6 = 90; //base
//For joints having two parellel servos, first preference is always given to right servo (if required)


double s_roll, s_pitch, s_yaw;
double e_roll, e_pitch, e_yaw;
double base_v, shoulder_v, elbow_v;
char c;
char temp[10];

void setup(){
Serial.begin(9600);
servo[0].attach(45);
servo[1].attach(44);
servo[2].attach(4);
servo[3].attach(5);
servo[4].attach(6);
servo[5].attach(7);
servo[6].attach(8);
servo[7].attach(9);
servo[8].attach(10);

servo[0].write(a1);
servo[1].write(a2);
servo[2].write(a2);
servo[3].write(a3);
servo[4].write(a3);
```

```
servo[5].write(a4);
servo[6].write(a4);
servo[7].write(a5);
servo[8].write(a6);

}

void loop(){
if(Serial.available()){
// Values Input from ROS::arm_data

temp = Serial.read();
s_roll = getValue()
temp = Serial.read();
s_pitch = getValue()
temp = Serial.read();
s_yaw = getValue()
temp = Serial.read();
e_roll = getValue()
temp = Serial.read();
e_pitch = getValue()
temp = Serial.read();
e_yaw = getValue()

Serial.println("Reading values inside if()");
}

base_v = findBaseAngle(s_yaw);
shoulder_v = findShoulderAngle(s_roll);
elbow_v = findElbowAngle(e_roll);

Serial.println("Reading values");
Serial.println(base_v);
Serial.println(shoulder_v);
Serial.println(elbow_v);

// exception handling
base_v = (base_v < 150)?base_v:150;
shoulder_v = (shoulder_v < 150)?shoulder_v:150;
elbow_v = (elbow_v < 150)?elbow_v:150;

servo[0].write(a1 + base_v);
servo[1].write(a2 + shoulder_v);
servo[2].write(a2 + shoulder_v);
servo[3].write(a3 + elbow_v);
servo[4].write(a3 + elbow_v);
servo[5].write(a4);
servo[6].write(a4);
servo[7].write(a5);
servo[8].write(a6);
```

```
}

double findBaseAngle(double yaw)
{
return yaw;
}

double findShoulderAngle(double roll)
{
return roll;
}

double findElbowAngle(double roll)
{
return roll;
}

double getValue()
{
int i=0;
while(Serial.available()){
temp[i] = Serial.read();
i++;
if(temp[i] == '\n')
break;
}
return atof(temp);
}
```

APPENDIX B : ROS Code

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <tf/transform_listener.h>
#include "iostream"
#include <fstream>

#include "std_msgs/MultiArrayLayout.h"
#include "std_msgs/MultiArrayDimension.h"
#include "std_msgs/Float64MultiArray.h"

using namespace std;



int main(int argc, char **argv)
{
  /*
  Must call ros::init always. 'init' needs to see argc & argv for some reaseon.
  The third argument to init() is the name of the node.
  */
  ros::init(argc, argv, "arm_data");

  /*
NodeHandle is the main access point to communications with the ROS system.
The first NodeHandle constructed will fully initialize this node, and the last
NodeHandle destructed will close down the node.
*/
  ros::NodeHandle node;

  /*
  The tf package provides an implementation of a TransformBroadcaster to help make the task of publishing transforms easier.
  To use the TransformBroadcaster, we need to include the tf/transform_broadcaster.h header file.
*/
  tf::TransformBroadcaster br;

  /*
  StampedTransfrom is a data strucrure of 'tf'
  Child class of Transform. Additional => time stamp
  */
  tf::StampedTransform transform_shoulder_prev, transform_shoulder_curr, transform_elbow_prev, transform_elbow_curr,
  transform_shoulder_diff, transform_elbow_diff;

  /*
  The tf package provides an implementation of a TransformListener to help make the task of receiving transforms easier.
  To use the TransformListener, we need to include the 'tf/transform_listener.h' header file.
  */
  tf::TransformListener listener;
```

```
double roll, pitch, yaw;

ros::Rate rate(2.0);

if(node.ok()){
        /*
                listener.lookupTransform("/frame1", "/frame2", ros::Time(0), StampedTransform_object);
                Query the listener for a specific transformation
                Parameters are:
                #1      We want the transform from this frame ...
                #2      ... to this frame.
                #3      The time at which we want to transform. Providing ros::Time(0) will just get us the latest available transform.
                #4      The object in which we store the resulting transform.
        */
        try{
                listener.lookupTransform("/openni_depth_frame", "/right_shoulder_1", ros::Time(0), transform_shoulder_curr);
                listener.lookupTransform("/openni_depth_frame", "/right_shoulder_1", ros::Time(0), transform_shoulder_prev);
                listener.lookupTransform("/openni_depth_frame", "/right_elbow_1", ros::Time(0), transform_elbow_curr);
                listener.lookupTransform("/openni_depth_frame", "/right_elbow_1", ros::Time(0), transform_elbow_prev);
        }
  catch (tf::TransformException ex){
     ROS_ERROR("%s",ex.what());
  }
}

while (node.ok()){

  try{
        listener.lookupTransform("/openni_depth_frame", "/right_shoulder_1", ros::Time(0), transform_shoulder_curr);
        listener.lookupTransform("/openni_depth_frame", "/right_elbow_1", ros::Time(0), transform_elbow_curr);
  }
  catch (tf::TransformException ex){
     ROS_ERROR("%s",ex.what());
     ros::Duration(1.0).sleep();
  }



  const tf::Quaternion &quaternion_shoulder_diff = transform_shoulder_curr.getRotation();
  const tf::Quaternion &quaternion_elbow_diff = transform_elbow_curr.getRotation();

  // Sending data to file "/dev/ttyUSB1"
  ofstream outfile;
  outfile.open("/dev/ttyUSB1");

  // Extracting RPY from tf::quaternion
  tf::Matrix3x3 m(quaternion_shoulder_diff);
  m.getRPY(roll, pitch, yaw);

        outfile << roll << endl << pitch << endl << yaw <<endl;
```

```
    cout <<"SHOULDER VALUES"<<endl;
    cout <<"roll"<<"\t"<<"pitch"<<"\t"<<"yaw"<< endl;
    cout << roll <<"\t"<< pitch <<"\t"<< yaw << endl;

    // Extracting RPY from tf::quaternion
    m = tf::Matrix3x3 (quaternion_elbow_diff);
    m.getRPY(roll,pitch,yaw);

    outfile << roll << endl << pitch << endl << yaw <<endl;

    cout <<"ELBOW VALUES"<<endl;
    cout <<"roll"<<"\t"<<"pitch"<<"\t"<<"yaw"<< endl;
    cout << roll <<"\t"<< pitch <<"\t"<< yaw << endl;

    outfile.close();

    rate.sleep();

  }

  return 0;
}
```

9. REFERENCES

[1] Shamsheer Verma 'Hand Gestures Remote Controlled Robotic Arm', B.Tech (ICE), Bharati Vidyapeeth College Of Engineering, New Delhi
[2] https://github.com/eyantra/CS308_GESTURE-CONTROLLED-ROBOT_2012
[3] https://www.edgefx.in/mems-based-gesture-controlled-robot/
[4] https://en.wikipedia.org/wiki/Kinect#Kinect_for_Microsoft_Windows
[5] https://msdn.microsoft.com/en-us/library/hh438998.aspx
[6] https://dlnmh9ip6v2uc.cloudfront.net/Robotics/SpiderControllerDagu.pdf

Image Source:

Figure 1: NASA (http://www.nasa.gov/mission_pages/shuttle/rms_gallery.html)
Figure 2: elsi.e-yantra.org
Figure 5: Ripublication.com
Figure 6: Ripublication.com
Figure 9: https://cdn.sparkfun.com//assets/parts/7/3/9/5/11498-01a.jpg