A REPORT ON Modular and Open-source platform for Gesture controlled robotic arm and its simulation

Parth Tulapurkar
Department of Mechanical Engineering
Vishwakarma Institute of Technology
Pune, India
parth.tulapurkar18@vit.edu

Nikhil Rathod
Department of Mechanical Engineering
Vishwakarma Institute of Technology
Pune, India
nikhil.rathod18@vit.edu

Anuprita Kaple
Department of Mechanical Engineering
Vishwakarma Institute of Technology
Pune, India
Anuprita.Kaple18@vit.edu

Prajakta Shinde
Department of Mechanical Engineering
Vishwakarma Institute of Technology
Pune, India
Prajakta.shinde18@vit.edu

Abstract -This report attends to the designing, simulating, and controlling a robotic arm with motion detection sensors to mimic the movement of the user's hand. In the field of space exploration, robots have become an integral tool for executing unmanned operations in environments that are hostile to human beings. The basic objective of this research is to develop an open-source 'modular' package for a robotic arm with 'n' Degrees-of-Freedom (DOF) and develop a 6 DOF version of the same at low costs.

Keywords — Algorithm, IMU, Kinematics, ROS, Python, gestures.

I. Introduction

The purpose of this project is to develop an open-source repository containing software-based building blocks of the robotic arm with 'n' DOF and use this repository to design and manufacture a 6-DOF robotic arm. Thus, the main objective becomes to enable organizations to manufacture a low-cost, highly-efficient, and 3-D printable Human motion mimicking robotic arm.

The robotic arm can serve as a human-machine interface to execute basic industrial activities like deploying, maneuvering, and capturing payloads. Alternatively, the robotic arm can also be used on buggies and rovers to perform pre-programmed operations such as pick-and-place, drilling, and scouting operations. The robotic arm is intended to be controlled remotely using a wearable glove clad with a potentiometer and an Inertial Measurement Unit (IMU) sensor maintaining an RF/Bluetooth/WiFi communication with the arm. The robotic arm overcomes the mechanical restraints of the human hand and is capable of achieving joint angles that are not possible with human arms using predefined gestures. Since the robotic arm will be developed on a modular platform, manufacturing more of them with varying DOF or scale-sizes in the future is entirely possible on the same platform. The device will be most useful and valuable when performing relatively small tasks in hostile environments where dispatching of a human is too great of a risk.

.

The research study is engrossed on designing, manufacturing, and implementing the robotic arm having six degrees of freedom and controlling that robotic arm using a motion-detecting sensor(MPU-9250). The plan is to use two such motion processing units; one on the wearable glove, and the other on the end effector of the robotic arm. Such an arrangement can relay the expected orientation of the end effector to the robotic arm. On the end effector, an IR/proximity sensor is to be set up for grasp detection. This robotic arm will be electrically triggered by using high torque DC servo motors for achieving six motions one of which is rotational motion. These motors are controlled by a microcontroller programmed using embedded C-programming. The voltage distribution to the servo motors is determined by the potentiometer enclosure and is transmitted with appropriate scaling to the robotic arm. The scaling factor depends on the dimensions of the said robotic arm and is translated using an embedded C program in the microcontroller. The relative position of the end effector is computed using hard-coded forward and inverse kinematics from specified values of joint parameters. The kinematics program utilizes cartesian coordinates obtained from joint angles and computes the final position using kinematic equations and link transformation. The forward and inverse kinematic program with its ensuing matrix arithmetic is done using python 3. The intuitive control of the robotic arm means control of the same at positions unattainable by human hand. This is done by pre-defined gesture motions, the expected IMU data for which is also encoded in the microcontroller. In the cases of error in the motion of the robotic arm due to clashing geometries of the links or improper angle of the servos, a program for failsafe conditions is to be executed that enlists the possible conditions of failures and returns the robotic arm to a default position. Finally, the simulations of the robotic arm are done on the Robotic Operating system (ROS) where the 3-D model of the robotic arm is converted to Universal Robotics Description Format (URDF) and simulated for testing purposes. The material of the body parts uses 3-D printed PLA, which is lightweight and provides considerable strength. The use of metals will be kept as limited as possible, keeping in mind the cold welding phenomenon in space.

II. MATERIAL AND METHODOLOGY

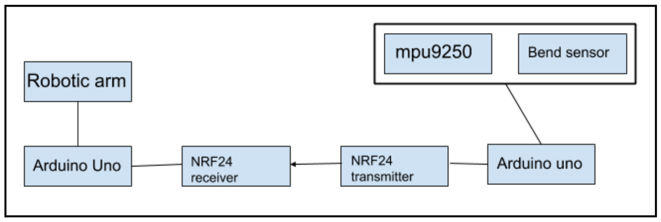
The Manipulation of Real-Time Kinect-Based Robotic Arm Using Double-Hand Gesture

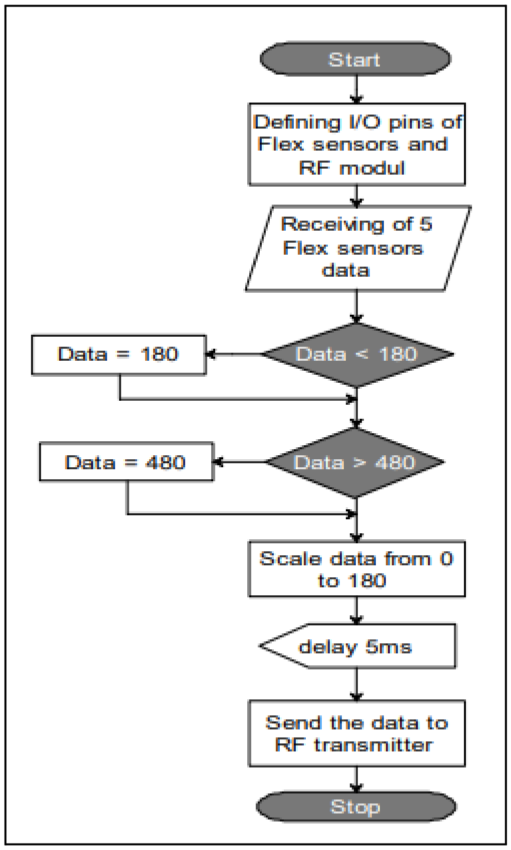
This project work is based on the works of the authors of the algorithm for determining the lengths of links of the said robotic arm. They laid out a framework for the algorithm in terms of computer vision which provided pathways for its implementation in various applications.

A. *Gesture Controlled Robotic Arm*

This project work is also based on the study of the development of mimicking robotic arm by IJST.

III. ALGORITHM





Start
Defining I/O pins of Flex sensors and RF modul
Receiving of 5 Flex sensors data
Data = 180
Data < 180
Data = 480
Data > 480
Scale data from 0 to 180
delay 5ms
Send the data to RF transmitter
Stop

Fig. No. 1: basic communication pipeline

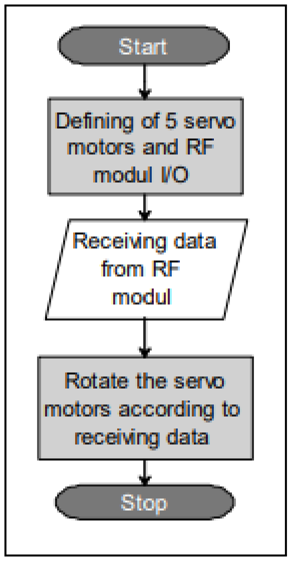

Start
Defining of 5 servo motors and RF modul I/O
Receiving data from RF modul
Rotate the servo motors according to receiving data
Stop

Fig. No. 2: basic algorithm pipeline

a. Microcontroller

While the main controller unit used is the Arduino UNO, other more powerful and compact microprocessors supporting embedded C will also work.

The Arduino Uno is a microcontroller board based on the ATmega328(datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the ATmega 16U2 programmed as a USB-to-serial converter.

b. MPU 9250

MPU-9250 is a multi-chip module (MCM) consisting of two dies integrated into a single QFN package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The MPU-9250 is a 9-axis Motion Tracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer, and a Digital Motion Processor (DMP) all in a small 3x3x1mm package available as a pin-compatible upgrade from the MPU6515. With its dedicated I2C sensor bus, the MPU-9250 directly provides complete 9-axis Motion Fusion output. MPU-9250 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I2C port. The input voltage is 3.3V and the weight is 15g.

While this IMU is inexpensive, it is used in many of today's electronics as this sensor is reliable for motion tracking and software integration. The drift, if any, is corrected using software filters; id est unscented Kalman filter, Magick, and Mahony filters. The IMU is used to teleoperate the robot's end-effector by intuitively controlling the wearer's arm. The intuitive controlling utilizes the roll, pitch, and yaw data from the glove which is scaled appropriately and transmitted to the robotic arm using an RF module (NRF28). The IMU is also used to determine the position of the end-effector relative to the base.

c. High torque servo motors

A servo motor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity, and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

JX-PDI-6225MG is the high torque servo motor selected for executing joint angles in the robotic arm. The main advantage of such a high torque is to reduce the latency time in overcoming the inertial of the links and end effector. It provides a 25 kg.cm torque with a maximum angle of 295°. The voltage range is 4.8 to 6.8V and the weight is 72g.

d. Potentiometer

A potentiometer (also known as a pot or pot-meter) is defined as a 3 terminal variable resistor in which the resistance is manually varied to control the flow of electric current. A potentiometer acts as an adjustable voltage divider. There are two main types of potentiometers:

1)Rotary potentiometer

2)Linear potentiometer

The potentiometer which is used to detect the movement here is the rotary potentiometer. There are many different uses of a potentiometer. The three main applications of a potentiometer are

1. Comparing the emf of a battery cell with a standard cell
2. Measuring the internal resistance of a battery cell
3. Measuring the voltage across a branch of a circuit

The proposed teleoperation system consists of three major subsystems: a custom-built glove worn by the user, a 6-DOF robotic arm, and a 2-fingered robotic claw mounted to the arm's end effector. The three subsystems are connected through ROS, which passes information across the whole system. The IMU-glove reads the user's hand position and orientation relative to their shoulder, as well as the orientation of the glove. The pose is converted to joint angles for the robotic arm. As the user moves their arm, the robotic arm, and the claw mirror the user's movements

e. Flex sensor

A flex sensor or bend sensor is a sensor that measures the amount of deflection or bending. Usually, the sensor is stuck to the surface, and resistance of the sensor element is varied by bending the surface. Since the resistance is directly proportional to the amount of bend it is used as a goniometer, and often called a flexible potentiometer.

Types:

1. Conductive ink based flex sensor
2. Fibre optic flex sensor
3. Capacitive flex sensor
4. Velostat flex sensor

IV. Design calculations and considerations

We designed the links and joints of the final assembly based on the proportions of the human hand so as to smoothly convert the human gesture into robotic movement. Because the repository is open-source there are no fixed dimensions of every individual link. However, we have considered a sample robotic arm for testing purposes.

We used the motion sensor to capture the coordinate position of each joint of the user [1] (the paper converted the coordinate system of the user into that of the robotic arm through a proportional conversion relationship.) The control system is mainly composed of the microprocessor, motion sensor, and the robotic arm.

Kinetic used to measure the coordinates of the shoulder (x_1',y_1',z_1') and arms (x_2',y_2',z_2'). The length of the link between the shoulder and elbow is calculated with the following formula:

$$D_1 = \sqrt{(x_2' - x_1')^2 + (y_2' - y_1')^2 + (z_2' - z_1')^2}.$$

If the Distance from shoulder to the elbow of the human arm is a. Then the proportionality between the lengths of 2 links is given by:

$$\frac{a_2}{D_1} = \frac{7}{\sqrt{(x_2' - x_1')^2 + (y_2' - y_1')^2 + (z_2' - z_1')^2}}.$$

In order to calculate the coordinate changes from the shoulder to the elbow of the robotic arm, we set the coordinate changes of the shoulder and elbow as (x_2'-x_1', y_2'-y_1', z_2'-z_1'), which we then multiply with the proportional relationship between the shoulder and elbow of the robotic arm. The result

$$\frac{a_2}{D_1} \bullet \left(x_2' - x_1', y_2' - y_1', z_2' - z_1'\right).$$

is the coordinate changes from the shoulder to the elbow of the robotic arm

:

If the coordinates of the elbow of the robotic arm are set to (0, 0, 7), while those of the elbow are(x_2,y_2,z_2). The coordinates of the shoulder of the users plus the coordinates changes from the shoulder to elbow of AL5D robotic arm resulted in the coordinates of the elbow of the robotic arm (x_2,y_2,z_2), as follows:

$$(x_2, y_2, z_2) = (0, 0, 7) + \frac{a_2}{D_1} \bullet \left(x_2' - x_1', y_2' - y_1', z_2' - z_1'\right).$$

After the coordinates (x_2,y_2,z_2) of the elbow of the robotic arm are calculated, for the purpose of the inverse kinematics calculation later, the coordinates of the wrist can be obtained. Thus, the elbow coordinates (x_3,y_3,z_3) of the robotic arm can be calculated. Set the wrist coordinates of the user as (x_2',y_2',z_2') and the elbow coordinates as (x_3',y_3',z_3'). The length of the link between the elbow and wrist can be calculated as follows:

$$D_2 = \sqrt{(x_3' - x_2')^2 + (y_3' - y_2')^2 + (z_3' - z_2')^2}.$$

The length of the link between the shoulder and elbow of the robotic arm was 18 cm. Hence, the proportional relationship a_3/D_2 between the lengths of the two links was as follows:

$$\frac{a_3}{D_2} = \frac{18.7325}{\sqrt{(x_3' - x_2')^2 + (y_3' - y_2')^2 + (z_3' - z_2')^2}}.$$

In order to calculate the coordinate changes from the elbow to the wrist of the robotic arm, we set the coordinate changes of the elbow and wrist of the user as (x_3'-x_2', y_3'-y_2', z_3'-z_2'), which are then multiplied with the proportional relationship between the elbow and wrist of the robotic arm. The result is the coordinate changes from the elbow to the wrist of the robotic arm:

$$\frac{a_3}{D_2} \bullet \left(x_3' - x_2', y_3' - y_2', z_3' - z_2'\right)$$

The coordinates of the elbow of the AL5D robotic arm are set to (x_2,y_2,z_2), while those of the wrist are (x_3,y_3,z_3). The coordinates of the elbow of the users (x_2,y_2,z_2) plus the coordinates changes from the elbow to the wrist of the robotic arm resulted in the coordinates of the wrist of the robotic arm (x_3,y_3,z_3), as follows:

$$(x_3, y_3, z_3) = (x_2, y_2, z_2) + \frac{a_3}{D_2} \bullet \left(x_3' - x_2', y_3' - y_2', z_3' - z_2'\right)$$

After the coordinates (x_3,y_3,z_3) of the elbow of the robotic arm were calculated, for the purpose of the inverse kinematics calculation later, the coordinates of the wrist should be obtained. Thus, the coordinates (x_4,y_4,z_4) of the end of the palm of the robotic arm should be calculated. Set the wrist coordinates of the user as (x_3',y_3',z_3') and the coordinates of the end of the palm as (x_4',y_4',z_4'). The length D3 of the link between the wrist and end of palm was calculated as follows:

$$D_3 = \sqrt{(x_4' - x_3')^2 + (y_4' - y_3')^2 + (z_4' - z_3')^2}.$$

The length of the link between the wrist and the end of the palm of the robotic arm was, in general 10 cm. Hence, the proportional relationship a_4/D_3 between the lengths of the two links was as follows:

$$\frac{a_4}{D_3} = \frac{10.0076}{\sqrt{(x_4' - x_3') + (y_4' - y_3') + (z_4' - z_3')}}.$$

In order to calculate the coordinate changes from the wrist to the end of the palm of the robotic arm, we set the coordinate changes of the wrist and the end of the palm of the user as (x_4'-x_3', y_4'-y_3', z_4'-z_3'), which is then multiplied with the proportional relationship between the wrist and the end of the palm of the robotic arm. The result was the coordinate changes from the wrist to the end of the palm of the robotic arm:

$$\frac{a_4}{D_3} \bullet \left(x_4' - x_3', y_4' - y_3', z_4' - z_3'\right).$$

$$(x_4, y_4, z_4) = (x_3, y_3, z_3) + \frac{a_4}{D_3} \bullet \left(x_4' - x_3', y_4' - y_3', z_4' - z_3'\right).$$

The coordinates of the wrist of the robotic arm are set to (x_3,y_3,z_3), while those of the end of the palm are (x_4,y_4,z_4). The coordinates of the wrist of the users (x_3,y_3,z_3) plus the coordinates changes from the wrist to the end of the palm of the robotic arm resulted in the coordinates of the end of the palm of the robotic arm (x_4,y_4,z_4), as follows:

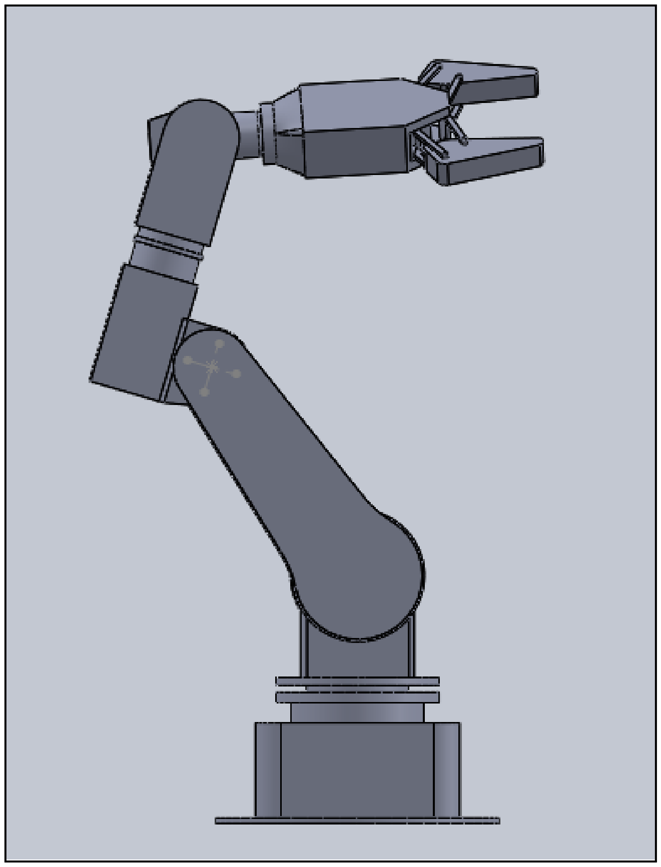


Fig. No. 3:final assembly based on calculated coordinates of joints

The final specifications are as follows:

Expected Mass: <=3.0 KGs

Dimensions: an end to end height = 69.48 CM

Maximum width of a link = 21.996 cm

Maximum breadth of a link = 12.496 cm

Estimated volume = 0.0535 cubic meter.

Base dimensions: 24.22 * 24.22 * 12.5 cm

L1 = Length of link 1 = 21.996 cm

L2 = Length of link 2 = 16 cm

L3 = Length of link 3 = 12 cm

Mass = 2.985 kilograms

Volume = 41 cubic centimeters

Surface area = 159 square meters

Center of mass: (centimeters)

X = -0.51

Y = 0.00

Z = 1.77

Principal axes of inertia and principal moments of inertia: Taken at the center of mass.

Ix = (-0.70, 0.71, -0.07)	Px = 0.90
Iy = (-0.71, -0.70, -0.05)	Py = 3.48
Iz = (-0.09, 0.02, 1.00)	Pz = 4.19

Moments of inertia:

Lxx = 2.24	Lxy = -1.28	Lxz = 0.19
Lyx = -1.28	Lyy = 2.16	Lyz = -0.15
Lzx = 0.19	Lzy = -0.15	Lzz = 4.17

Moments of inertia Taken at the output coordinate system.

Ixx = 95.42	Ixy = -1.29	Ixz = -26.84
Iyx = -1.29	Iyy = 103.19	Iyz = -0.12
Izx = -26.84	Izy = -0.12	Izz = 12.01

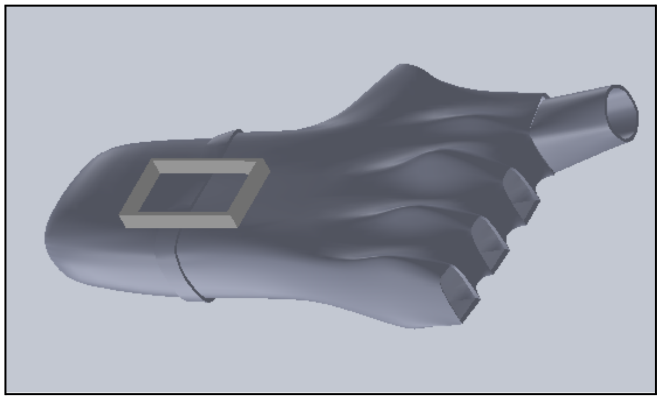


Fig. No. 4: Tentative design of gesture control glove

The glove includes a housing for the microcontroller on the top part and flex sensors for sensing the bending of the user's fingers. The movement of individual fingers will guide the movement of the robotic arm when any further movement by the user is out of scope of the human arm.

V. IMU CALIBRATION AND GESTURE CONTROL

A. Theory

The final position and the orientation of the end effector is decided by the IMUs on the glove of the user. This orientation is determined by the sensor fusion of the accelerometer, gyroscope, and the magnetometer present on the IMU. This sensor fusion is called Altitude Heading Reference System (AHRS).

Orientation is how far the object has been rotated wrt some reference frame. To define orientation, we need

1. Reference frame
2. Specifying of rotation

For example, roll, pitch, yaw(reference: local horizon, rotation: angular deviation)

Rotation can also be defined by direction cosine matrices (DCM) or quaternion. These

Two represent a 3-dimensional rotation between two coordinate frames.

We can find an absolute orientation of a device at rest using an accelerometer, magnetometer, and a gyroscope using the device's body coordinate frame relative to the local north, east, and down coordinate.
When at rest, the accelerometer measures the device's acceleration due to gravity, and the direction of this acceleration is 'up'. The magnetometer measures the direction of the magnetic north, which can move up or down of the north-east plane due to geographical location.

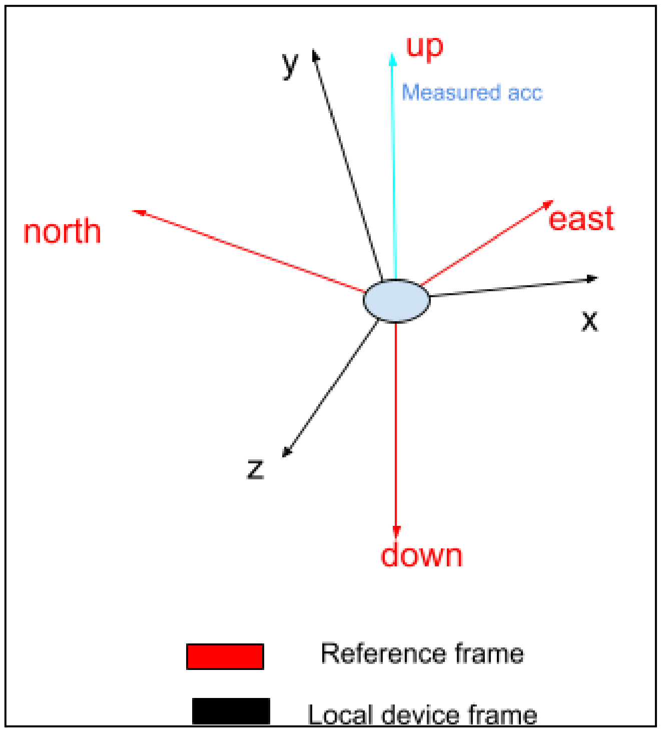


Fig. No. 5:global and local device reference frames for IMU

To correct for this deflection, we need to apply cross-products. Since the sensor returns directional vectors, the cross product of the down direction and the measured magnetic field is east. Similarly, the north is the cross product of the east and down. Direction cosines can be built from the north, east, and down vectors.
Accelerometers measure acceleration in all directions so that when a system is moving, the down direction gets an offset with the motion. Also if the accelerometers are not located at the COG of the system any rotational motion will change the linear acceleration vectors; even if the system has none and is simply rotating along its axis. To solve this, we provide a threshold for the max and min values in terms of g that are not to be exceeded, else the device ignores it. Or add the gyroscope to the combination of systems, incorporating the dead reckoning essentially making it an IMU.

For calibration, let max_X, max_Y, max_Z, and min_X, min_Y, min_Z be the maximum and minimum magnitudes of vectors recorded by the accelerometer. For a device at rest, it can have 6 possible attitudes

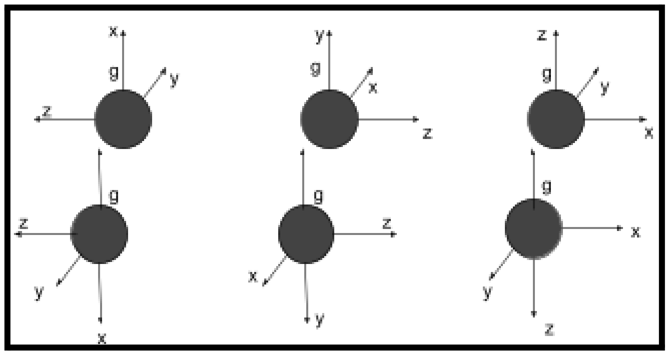


Fig. No. 6: possible orientations of IMU reference frames

Once that we have all 6 values, we have to do some basic calculations to get the offsets and scales. Accelerometer calibration is done using these two values (per axis).For offsets:

$$offset_x = \frac{min_x + max_x}{2}$$

$$offset_y = \frac{min_y + max_y}{2}$$

$$offset_z = \frac{min_z + max_z}{2}$$

For scales,

$$scale_x = \frac{1}{max_x - offset_x}$$

$$scale_y = \frac{1}{max_y - offset_y}$$

$$scale_z = \frac{1}{max_z - offset_z}$$

Once we have the offsets and scales, we have to just apply them to our accelerometer measurement. This just squeezes or expands the measurements and adds an offset.

$$calibrated_x = (acc_x - offset_x)scale_x$$

$$calibrated_y = (acc_y - offset_y)scale_y$$

$$calibrated_z = (acc_z - offset_z)scale_z$$

Magnetometers are affected by disturbances in the magnetic field. If this magnetic field is part of the system, and rotates with the magnetometer then it can be calibrated out.
These disturbances are caused by two kinds of hardware; the hard iron source and the soft iron source.
Hard iron sources are stuff that generates their own magnetic field like permanent magnets and device circuitry. When measuring the earth's magnetic field, the hard iron magnetic field would contribute to the resulting measurement and when rotating the system in a single axis while still measuring the field, the result would be a circle that is offset from the origin.

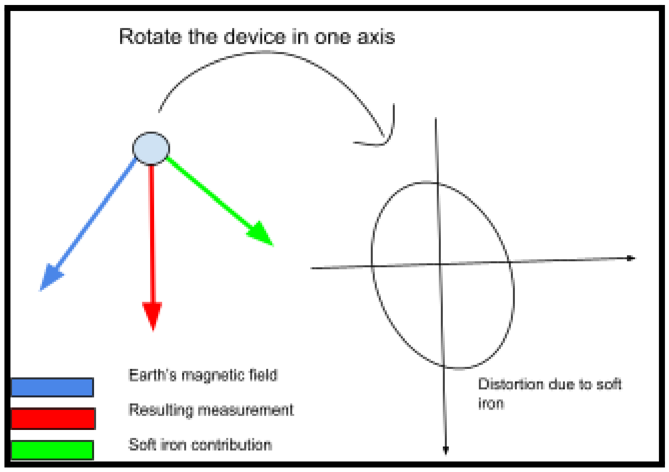


Fig. No. 7:circle offset method of magnetometer calibration

Soft iron sources do not have their own magnetic fields but interact with external fields to create disturbances.
If the device is rotated in all of the 4 pi radian directions without any of these sources, then it would be creating a perfect sphere with the radius as the magnitude of the field. When distorted and shifted by soft and hard sources, the resulting shape is an irregular spheroid. To correct the spheroid back to the sphere, so as to calibrate it, we need to transform the received data matrix with the hard and soft iron bias.

$X_{corrected}$ = (X.b).A
Where b = (3x1) vector, hard iron bias
A = (3x3) matrix, soft iron bias

$$Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exz + 2Fyz + 2Gx + 2Hy + 2Iz$$

For the soft iron core, the equation of an ellipsoid is

Here A, B,C,D,E,F,G,H, and I are ellipsoid parameters
With the following expression we can do this transformation. Note that c is the center of the transformation and e_{xy} is the XY element of the transformation matrix.
Mag_x, mag_y, and mag_z are magnetic field vectors from sensors.

$$\begin{pmatrix} mag_x \\ max_y \\ max_z \end{pmatrix} = \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} \begin{pmatrix} mag_x - c_x \\ max_y - c_y \\ mag_z - c_z \end{pmatrix}$$

B. Algorithm

To accurately map the Roll, pitch, yaw (RPY) coordinates from the IMU, we use a predictive model for better motion of the end effector. This predictive model is called Kalman filter. Kalman filter is a mathematical prediction model used to determine the state of an object in 3-D space. This filter was used onboard the computer of Apollo 11 to map the trajectory. Its algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real-time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required. This process is repeated at every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state.

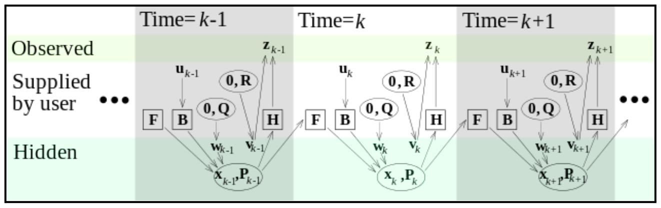
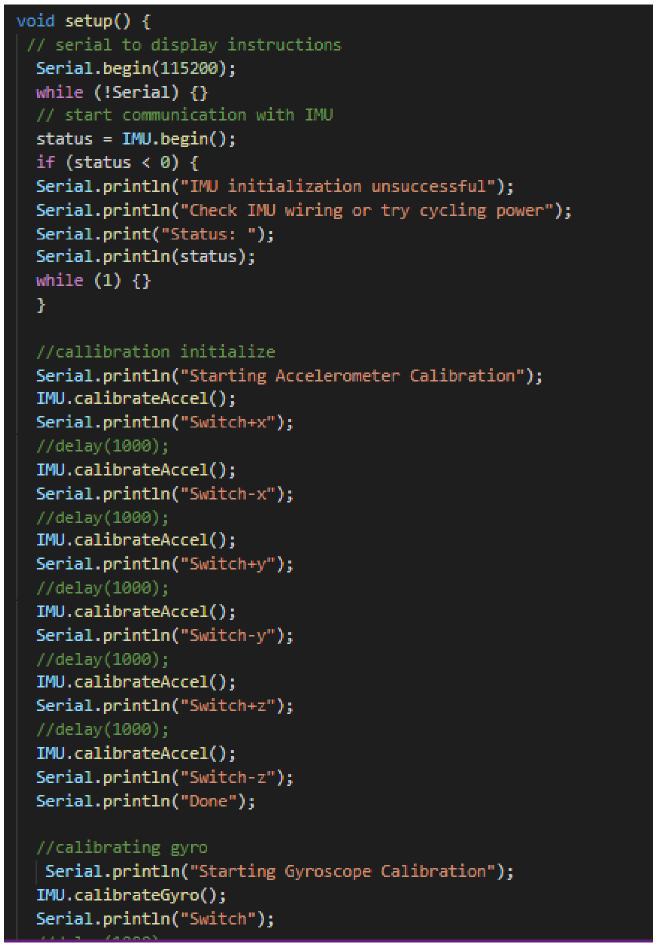
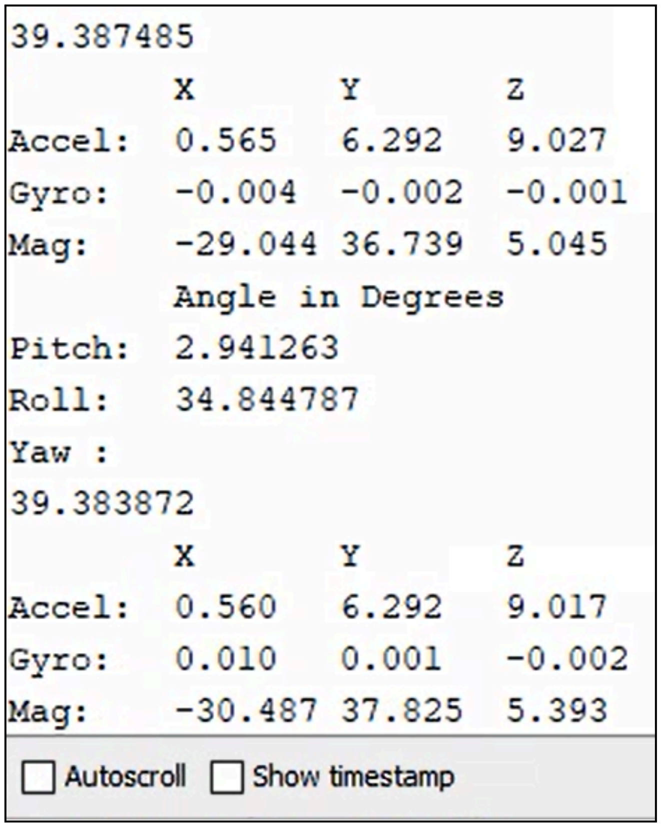


Fig. No. 8:Kalman filter algorithm



```
void setup() {
 // serial to display instructions
  Serial.begin(115200);
  while (!Serial) {}
  // start communication with IMU
  status = IMU.begin();
  if (status < 0) {
  Serial.println("IMU initialization unsuccessful");
  Serial.println("Check IMU wiring or try cycling power");
  Serial.print("Status: ");
  Serial.println(status);
  while (1) {}
  }

  //callibration initialize
  Serial.println("Starting Accelerometer Calibration");
  IMU.calibrateAccel();
  Serial.println("Switch+x");
  //delay(1000);
  IMU.calibrateAccel();
  Serial.println("Switch-x");
  //delay(1000);
  IMU.calibrateAccel();
  Serial.println("Switch+y");
  //delay(1000);
  IMU.calibrateAccel();
  Serial.println("Switch-y");
  //delay(1000);
  IMU.calibrateAccel();
  Serial.println("Switch+z");
  //delay(1000);
  IMU.calibrateAccel();
  Serial.println("Switch-z");
  Serial.println("Done");

  //calibrating gyro
   Serial.println("Starting Gyroscope Calibration");
  IMU.calibrateGyro();
  Serial.println("Switch");
```

Fig. No. 9:IMU calibration and Kalman filter code for data retrieval



```
39.387485
        X       Y       Z
Accel:  0.565   6.292   9.027
Gyro:   -0.004  -0.002  -0.001
Mag:    -29.044 36.739  5.045
        Angle in Degrees
Pitch:  2.941263
Roll:   34.844787
Yaw :
39.383872
        X       Y       Z
Accel:  0.560   6.292   9.017
Gyro:   0.010   0.001   -0.002
Mag:    -30.487 37.825  5.393
```

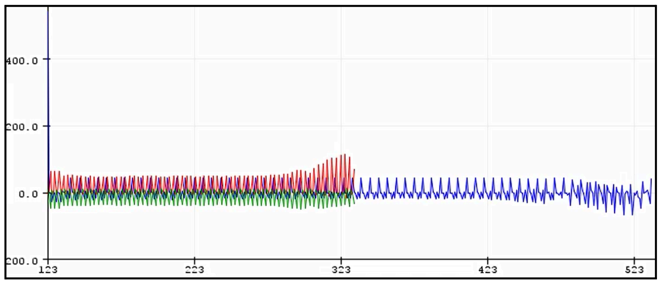


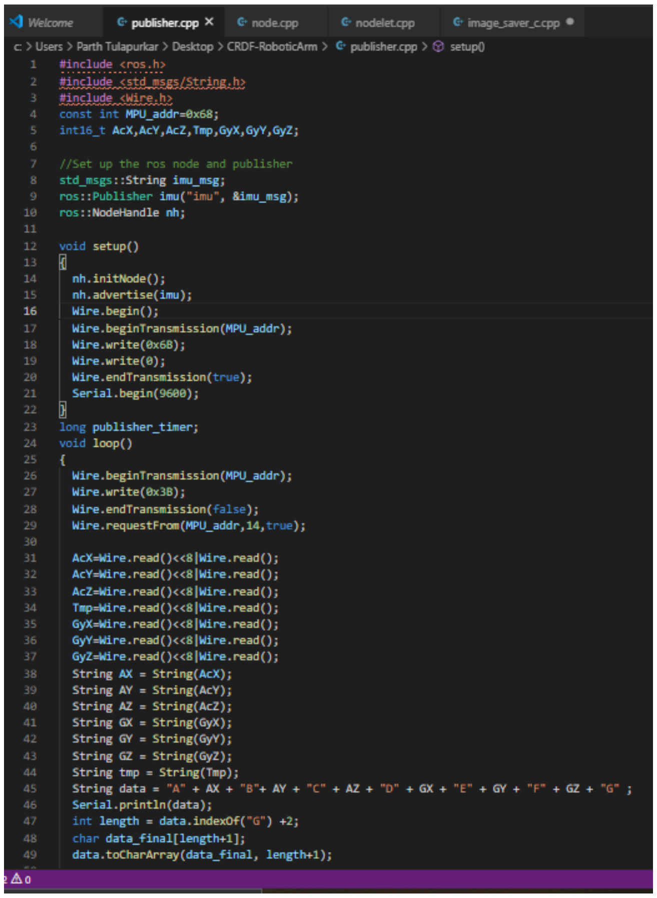
Fig. No. 9:IMU calibration and Kalman filter output for data retrieval

C. IMU-ROS interface and circuit simulation

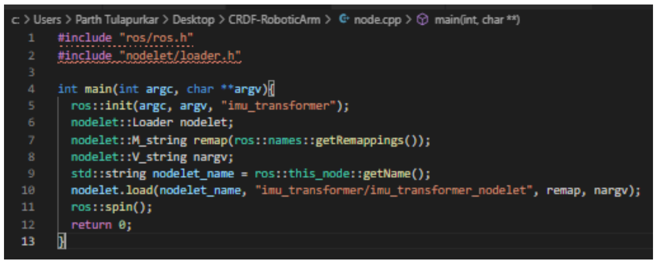
A node, according to formal ROS documentation, "Is a process that performs computations. Nodes are combined together into a graph and communicate with one another using streaming topics … a robot control system will usually comprise many nodes." A node can be a publisher (exports data) or a subscriber (imports data). In this case, the Arduino board acts as a publisher node. It publishes sensor readings as a string.After getting the sensor's readings, we start looking for a suitable message type, and the easiest way to send this data to ROS, is to use the string message type in ROS by converting the readings into a string type then concatenating all the readings into a string message. We find the *sensor_msgs* messages type in the ROS system, and part of it can hold the IMU's numeric values of sensors readings.For example:

“A55B230C-100D-200E-600F450G”

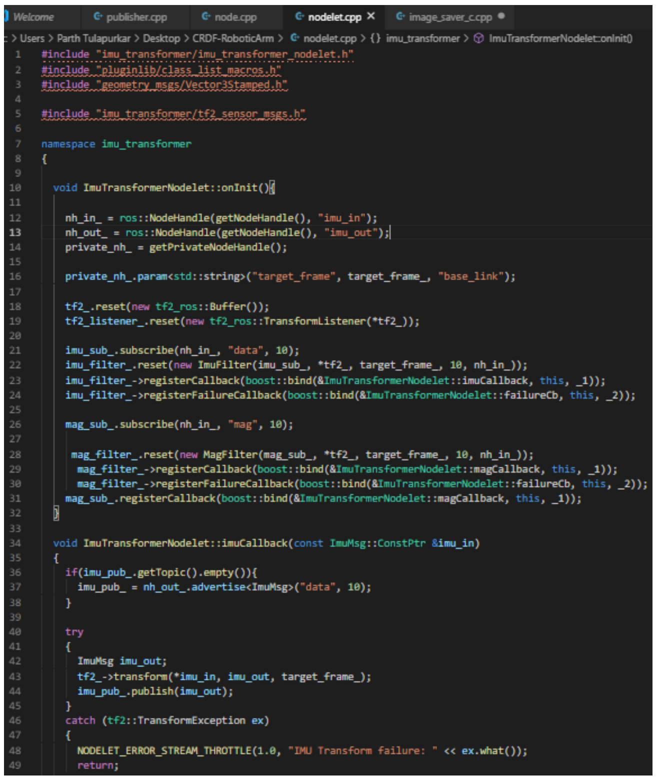
Here, one can easily extract each value and convert it -in the subscriber node- into integers back again.
We also make sub-nodes for individual IMUs on the glove so that each IMU has one to one communication with the end effector.


Welcome
publisher.cpp
node.cpp
nodelet.cpp
image_saver_c.cpp
c: > Users > Parth Tulapurkar > Desktop > CRDF-RoboticArm > publisher.cpp > setup()

```cpp
#include <ros.h>
#include <std_msgs/String.h>
#include <Wire.h>
const int MPU_addr=0x68;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

//Set up the ros node and publisher
std_msgs::String imu_msg;
ros::Publisher imu("imu", &imu_msg);
ros::NodeHandle nh;

void setup()
{
  nh.initNode();
  nh.advertise(imu);
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
}
long publisher_timer;
void loop()
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);

  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();
  Tmp=Wire.read()<<8|Wire.read();
  GyX=Wire.read()<<8|Wire.read();
  GyY=Wire.read()<<8|Wire.read();
  GyZ=Wire.read()<<8|Wire.read();
  String AX = String(AcX);
  String AY = String(AcY);
  String AZ = String(AcZ);
  String GX = String(GyX);
  String GY = String(GyY);
  String GZ = String(GyZ);
  String tmp = String(Tmp);
  String data = "A" + AX + "B"+ AY + "C" + AZ + "D" + GX + "E" + GY + "F" + GZ + "G" ;
  Serial.println(data);
  int length = data.indexOf("G") +2;
  char data_final[length+1];
  data.toCharArray(data_final, length+1);
```


c: > Users > Parth Tulapurkar > Desktop > CRDF-RoboticArm > node.cpp > main(int, char **)

```cpp
#include "ros/ros.h"
#include "nodelet/loader.h"

int main(int argc, char **argv){
  ros::init(argc, argv, "imu_transformer");
  nodelet::Loader nodelet;
  nodelet::M_string remap(ros::names::getRemappings());
  nodelet::V_string nargv;
  std::string nodelet_name = ros::this_node::getName();
  nodelet.load(nodelet_name, "imu_transformer/imu_transformer_nodelet", remap, nargv);
  ros::spin();
  return 0;
}
```


Welcome
publisher.cpp
node.cpp
nodelet.cpp
image_saver_c.cpp
> Users > Parth Tulapurkar > Desktop > CRDF-RoboticArm > nodelet.cpp > {} imu_transformer > ImuTransformerNodelet::onInit()

```cpp
#include "imu_transformer/imu_transformer_nodelet.h"
#include "pluginlib/class_list_macros.h"
#include "geometry_msgs/Vector3Stamped.h"

#include "imu_transformer/tf2_sensor_msgs.h"

namespace imu_transformer
{

  void ImuTransformerNodelet::onInit(){

    nh_in_ = ros::NodeHandle(getNodeHandle(), "imu_in");
    nh_out_ = ros::NodeHandle(getNodeHandle(), "imu_out");
    private_nh_ = getPrivateNodeHandle();

    private_nh_.param<std::string>("target_frame", target_frame_, "base_link");

    tf2_.reset(new tf2_ros::Buffer());
    tf2_listener_.reset(new tf2_ros::TransformListener(*tf2_));

    imu_sub_.subscribe(nh_in_, "data", 10);
    imu_filter_.reset(new ImuFilter(imu_sub_, *tf2_, target_frame_, 10, nh_in_));
    imu_filter_->registerCallback(boost::bind(&ImuTransformerNodelet::imuCallback, this, _1));
    imu_filter_->registerFailureCallback(boost::bind(&ImuTransformerNodelet::failureCb, this, _2));

    mag_sub_.subscribe(nh_in_, "mag", 10);

     mag_filter_.reset(new MagFilter(mag_sub_, *tf2_, target_frame_, 10, nh_in_));
      mag_filter_->registerCallback(boost::bind(&ImuTransformerNodelet::magCallback, this, _1));
      mag_filter_->registerFailureCallback(boost::bind(&ImuTransformerNodelet::failureCb, this, _2));
    mag_sub_.registerCallback(boost::bind(&ImuTransformerNodelet::magCallback, this, _1));
  }

  void ImuTransformerNodelet::imuCallback(const ImuMsg::ConstPtr &imu_in)
  {
    if(imu_pub_.getTopic().empty()){
      imu_pub_ = nh_out_.advertise<ImuMsg>("data", 10);
    }

    try
    {
      ImuMsg imu_out;
      tf2_->transform(*imu_in, imu_out, target_frame_);
      imu_pub_.publish(imu_out);
    }
    catch (tf2::TransformException ex)
    {
      NODELET_ERROR_STREAM_THROTTLE(1.0, "IMU Transform failure: " << ex.what());
      return;
```

Fig. No. 10(a): a publisher, node and nodelet pertaining to IMU data collection for ROS

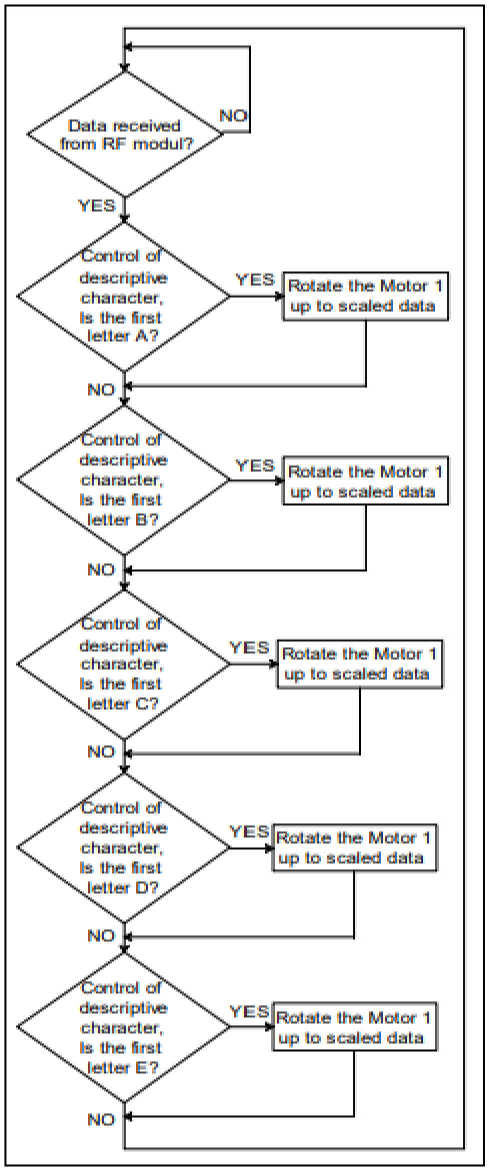

Data received from RF modul?
NO
YES
Control of descriptive character, Is the first letter A?
YES
Rotate the Motor 1 up to scaled data
NO
Control of descriptive character, Is the first letter B?
YES
Rotate the Motor 1 up to scaled data
NO
Control of descriptive character, Is the first letter C?
YES
Rotate the Motor 1 up to scaled data
NO
Control of descriptive character, Is the first letter D?
YES
Rotate the Motor 1 up to scaled data
NO
Control of descriptive character, Is the first letter E?
YES
Rotate the Motor 1 up to scaled data
NO

Fig. No. 10(b): a publisher and subscriber node flow diagram

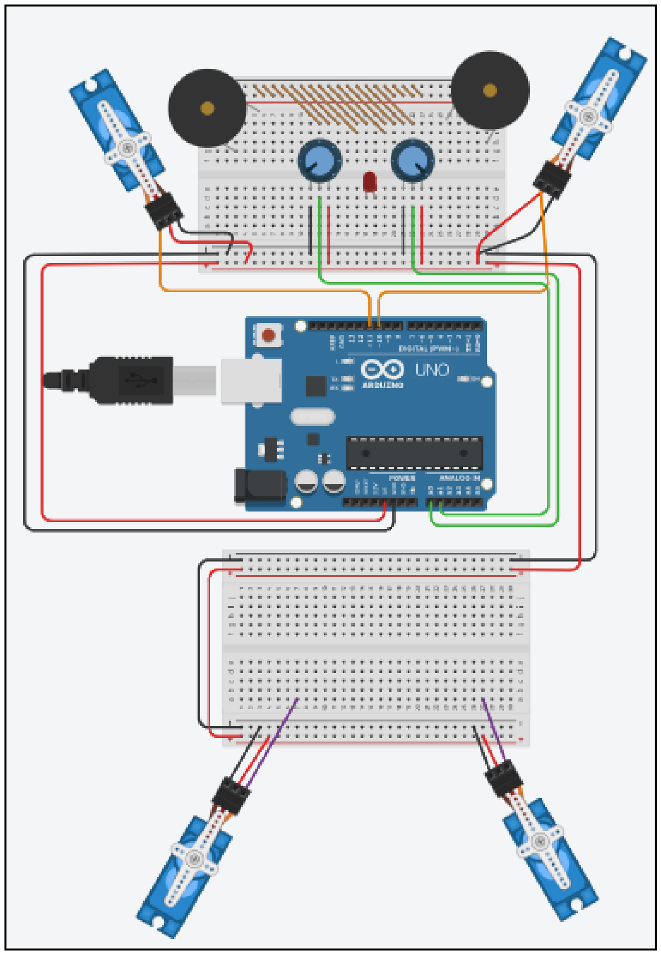


Fig. No. 11:Tinker cad simulation for controlling the servos via IMU. Here ultrasound sensors represent imu and their corresponding outputs represent Roll and yaw of the glove.

VI. Kinematics

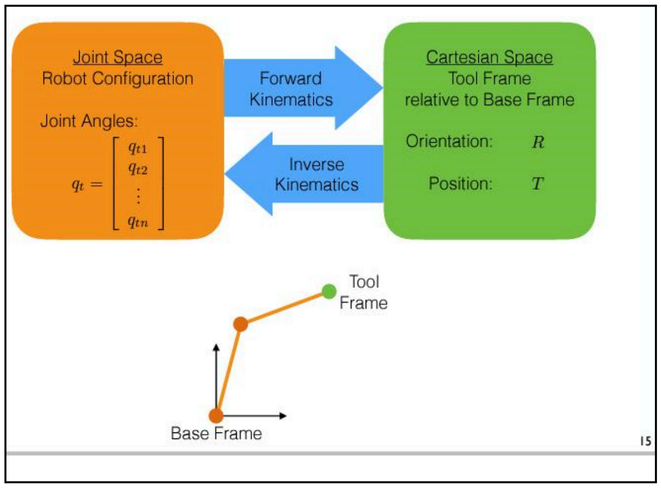


Fig. No. 11:solving joint space and cartesian space using forward and inverse kinematics

Forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters. The reverse process that computes the joint parameters that achieve a specified position of the end-effector is known as inverse kinematics.

Inverse kinematics is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain, such as a robot manipulator or animation character's skeleton, in a given position and orientation relative to the start of the chain. Given joint parameters, the position and orientation of the chain's end, e.g. the hand of the character or robot, can typically be calculated directly using multiple applications of trigonometric formulas, a process known as forward kinematics.

Algorithm for forward kinematics -
Takes array of joint angles (q)
Returns transformation matrix from robot base to end end-effector.
Initialize final transformation matrix as 4x4 identity matrix.
Start for loops for number of links,get roll, pitch, yaw for ith joint.
Get x, y, z translation and axis of rotation for ith joint.
Compute homogeneous matrix from rpy and xyz translation.
Compute homogeneous angleAxis matrix from axis and rotation angle (qi) for the ith joint. (Assume x,y,z translation = 0)
Compute total homogeneous matrix as a product of above matrices.Multiply total homogeneous matrix with final transformation matrix.Return final transformation matrix.

Gimbal lock is the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space.

Unlike forward kinematics, the complexity in the inverse kinematics of industrial robot manipulators arises from their geometry and nonlinear equations (trigonometric equations) occurring between Cartesian space and joint space.
Some other difficulties in inverse kinematics problem are:
1.kinematic equations are coupled,
2.multiple solutions and singularities may exist.
Mathematical solutions for inverse kinematics problems may not always correspond to physical solutions and the method of its solution depends on the robot configuration.
Conventional numerical approaches to the inverse calibration of robots are time-consuming and suffer from numerical problems of ill-conditioning and singularities.

Inverse kinematics can be solved by 2 approaches either by analytically (in closed form) or by numerically (in iterative form).
Analytical approach preferred for 6dof arm with sufficient conditions that -3 consecutive rotational joint axes are incident (e.g., spherical wrist), or 3 consecutive rotational joint axes are parallel and this method also provides systematic ways for generating a reduced set of equations to be solved.
While numerical approach is slower but easier to set up. in its basic form, it uses the(analytical) Jacobian matrix of the direct kinematics map

$$J_r(q) = \frac{\partial f_r(q)}{\partial q}$$

The Jacobian Transpose always exists, as opposed to the Jacobian Inverse. It is also less computationally expensive, which means that the performance will be faster.
Additionally, the Jacobian Transpose is easier to implement than the Jacobian Inverse.
A rough approximation to the Jacobian Inverse that works in many simple cases is replacing the Jacobian Inverse with the Jacobian Transpose.

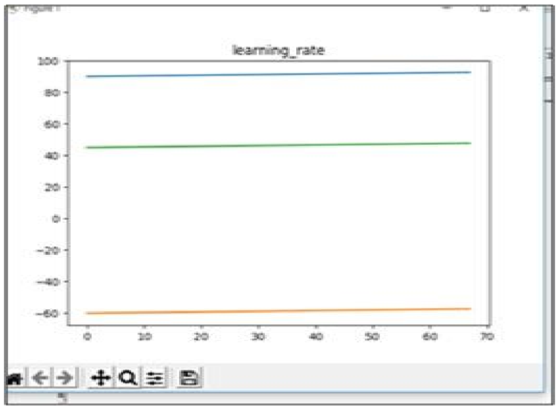


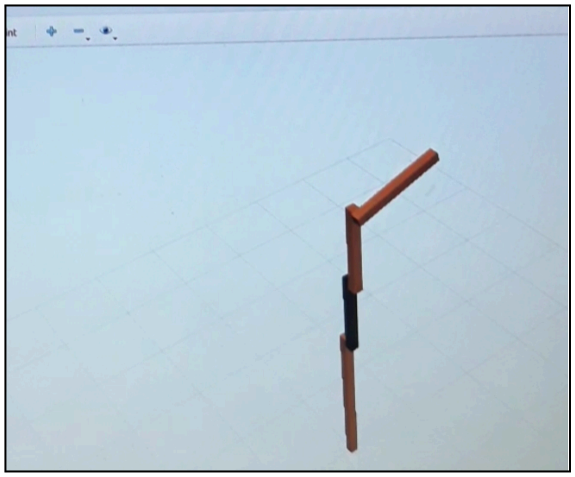
Fig : Inverse kinematics -Joint angle Vs Time graph

VII. ROS SETUP AND SIMULATION

Robot Operating System is robotics middleware (i.e. collection of software frameworks for robot software development). It provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management

Making use of Gazebo and Rviz, the various packages available online were tested to insure the correct installation of all the needed libraries of ROS, necessary to run the simulation. After a detailed study and comparison of the online packages available in Rviz and Gazebo, Rviz was chosen as the simulation platform.

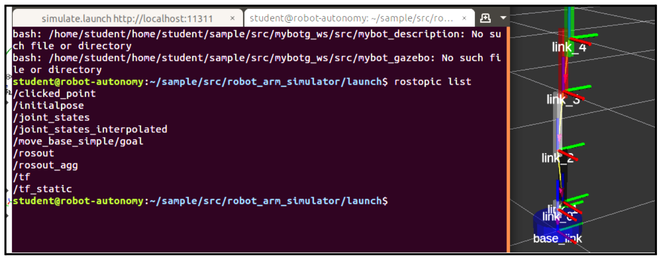
A detailed study of URDF and the other files needed to execute a simulation was carried out .Starting with a simple 3-link robotic arm , the logic of writing Urdf and launch file was developed. Further 4th link was added in the urdf and tested (visible in the pic).



A complete package containing launch file, urdf and config file was developed of a 4-link robotic arm but due to recurring errors even after many changes, we had to discard it.

Motion:

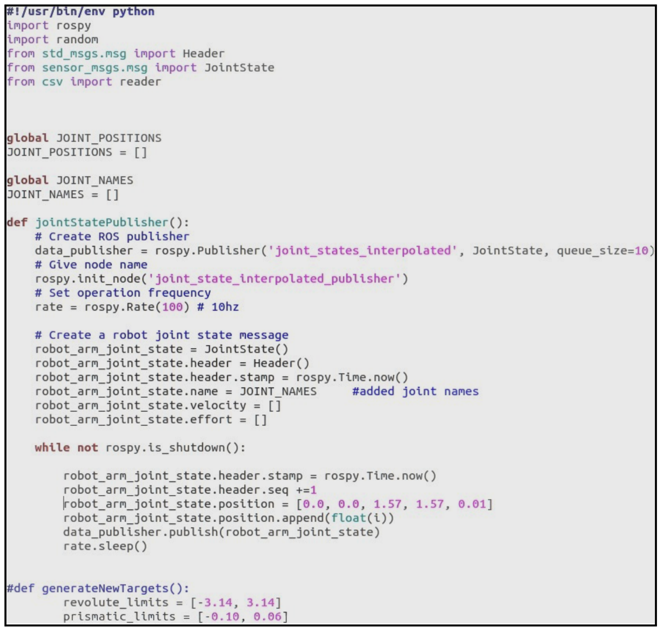
In the next step we tried to give motion to the robotic arm other than the default one that existed due to the JSP(joint state publisher).The packages use JSP to publish some fake joint state messages where values are taken from the sliders in the GUI. To move the robotic arm on our command, angles need to be published at the topic to which the robotic arm is subscribing to, for the joint angles. The various topics that were in use were studied using commands like rostopic list, rostopic echo, rostopic info joint_states in the terminal.



```
simulate.launch http://localhost:11311    student@robot-autonomy: ~/sample/src/ro...
bash: /home/student/home/student/sample/src/mybotg_ws/src/mybot_description: No su
ch file or directory
bash: /home/student/home/student/sample/src/mybotg_ws/src/mybot_gazebo: No such fi
le or directory
student@robot-autonomy:~/sample/src/robot_arm_simulator/launch$ rostopic list
/clicked_point
/initialpose
/joint_states
/joint_states_interpolated
/move_base_simple/goal
/rosout
/rosout_agg
/tf
/tf_static
student@robot-autonomy:~/sample/src/robot_arm_simulator/launch$
```

The existing topics when the robotic arm is simulated

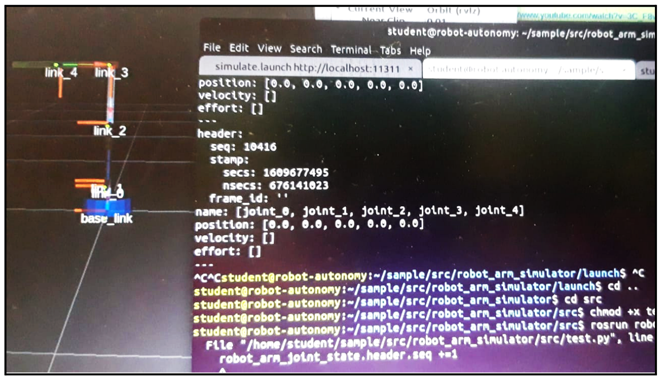
1)We considered and tried options like move_it and ros_control to give motion to the simulation arm. The needed format to give command from the terminal was:$ rostopic pub my_controller/robot/joint_states sensor_msgs/JointState '{header: {seq: 0, stamp: {secs: 0, nsecs: 0}, frame_id: ""}, name: ["the_urdf_joint_name"], position: [1.0], velocity: [0.0], effort: [0.0]}'
But the issue prevailing was that this command from terminal was publishing the angles only once whereas the command going by default from the joint-state-publisher had a high frequency. Due to this the movement corresponding to the terminal command couldnt be observed. Thus a python file was written to give continuous commands at a high frequency.



```python
#!/usr/bin/env python
import rospy
import random
from std_msgs.msg import Header
from sensor_msgs.msg import JointState
from csv import reader



global JOINT_POSITIONS
JOINT_POSITIONS = []

global JOINT_NAMES
JOINT_NAMES = []

def jointStatePublisher():
    # Create ROS publisher
    data_publisher = rospy.Publisher('joint_states_interpolated', JointState, queue_size=10)
    # Give node name
    rospy.init_node('joint_state_interpolated_publisher')
    # Set operation frequency
    rate = rospy.Rate(100) # 10hz

    # Create a robot joint state message
    robot_arm_joint_state = JointState()
    robot_arm_joint_state.header = Header()
    robot_arm_joint_state.header.stamp = rospy.Time.now()
    robot_arm_joint_state.name = JOINT_NAMES     #added joint names
    robot_arm_joint_state.velocity = []
    robot_arm_joint_state.effort = []

    while not rospy.is_shutdown():

        robot_arm_joint_state.header.stamp = rospy.Time.now()
        robot_arm_joint_state.header.seq +=1
        robot_arm_joint_state.position = [0.0, 0.0, 1.57, 1.57, 0.01]
        robot_arm_joint_state.position.append(float(i))
        data_publisher.publish(robot_arm_joint_state)
        rate.sleep()


#def generateNewTargets():
        revolute_limits = [-3.14, 3.14]
        prismatic_limits = [-0.10, 0.06]
```

The python file giving input to the robotic arm

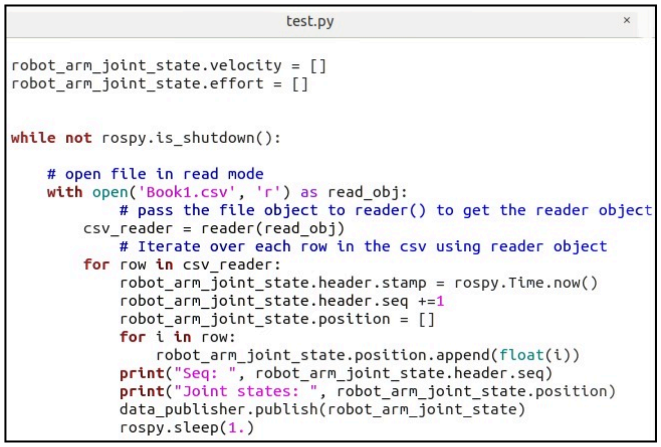
The file can be run using terminal .This file is publishing the input joint angles at topic 'joint_states_interpolated' . But by this only one position command is sent at a time. The robotic arm takes one position as per entered in the file. Using rostopic echo the published angles can be seen in another terminal.(visible in the pic)



```
student@robot-autonomy: ~/sample/src/robot_arm_sim
File  Edit  View  Search  Terminal  Tabs  Help
simulate.launch http://localhost:11311
position: [0.0, 0.0, 0.0, 0.0, 0.0]
velocity: []
effort: []
---
header:
  seq: 10416
  stamp:
    secs: 1609677495
    nsecs: 676141023
  frame_id: ''
name: [joint_0, joint_1, joint_2, joint_3, joint_4]
position: [0.0, 0.0, 0.0, 0.0, 0.0]
velocity: []
effort: []
---
^C^Cstudent@robot-autonomy:~/sample/src/robot_arm_simulator/launch$ ^C
student@robot-autonomy:~/sample/src/robot_arm_simulator/launch$ cd ..
student@robot-autonomy:~/sample/src/robot_arm_simulator$ cd src
student@robot-autonomy:~/sample/src/robot_arm_simulator/src$ chmod +x te
student@robot-autonomy:~/sample/src/robot_arm_simulator/src$ rosrun rob
  File "/home/student/sample/src/robot_arm_simulator/src/test.py", line
    robot_arm_joint_state.header.seq +=1
```

The arm after taking running the python file

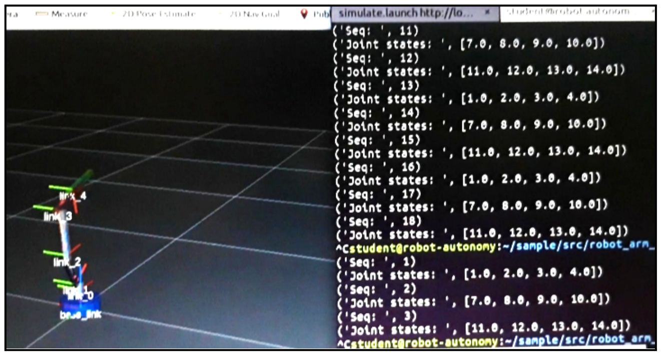
2)To give various commands a csv file is included which will have joint angles, input from the imu. By including a csv reader in the python file, the robotic arm gets command of various set of joint angles and moves accordingly.



```
robot_arm_joint_state.velocity = []
robot_arm_joint_state.effort = []


while not rospy.is_shutdown():

    # open file in read mode
    with open('Book1.csv', 'r') as read_obj:
            # pass the file object to reader() to get the reader object
        csv_reader = reader(read_obj)
            # Iterate over each row in the csv using reader object
        for row in csv_reader:
            robot_arm_joint_state.header.stamp = rospy.Time.now()
            robot_arm_joint_state.header.seq +=1
            robot_arm_joint_state.position = []
            for i in row:
                robot_arm_joint_state.position.append(float(i))
            print("Seq: ", robot_arm_joint_state.header.seq)
            print("Joint states: ", robot_arm_joint_state.position)
            data_publisher.publish(robot_arm_joint_state)
            rospy.sleep(1.)
```

The final python file with the csv reader

Due to the csv reader all the joint angles in the csv file are read and sent as input. The robotic arm takes various positions as per the various set of joint angles from the csv file. The published set of angles are also printed on the screen (visible in the pic)



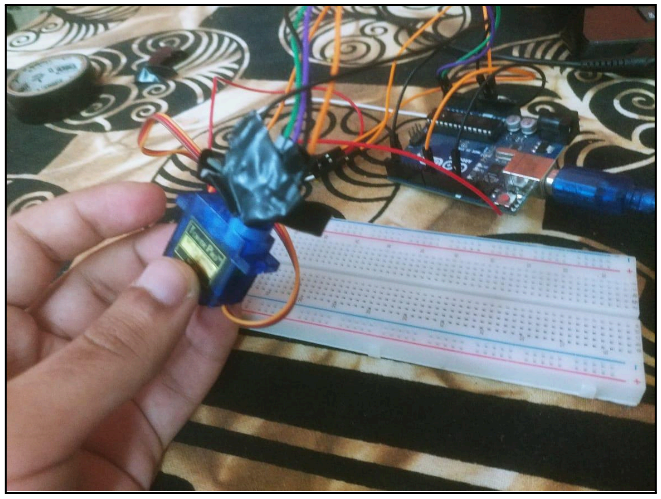
```
('Seq: ', 11)
('Joint states: ', [7.0, 8.0, 9.0, 10.0])
('Seq: ', 12)
('Joint states: ', [11.0, 12.0, 13.0, 14.0])
('Seq: ', 13)
('Joint states: ', [1.0, 2.0, 3.0, 4.0])
('Seq: ', 14)
('Joint states: ', [7.0, 8.0, 9.0, 10.0])
('Seq: ', 15)
('Joint states: ', [11.0, 12.0, 13.0, 14.0])
('Seq: ', 16)
('Joint states: ', [1.0, 2.0, 3.0, 4.0])
('Seq: ', 17)
('Joint states: ', [7.0, 8.0, 9.0, 10.0])
('Seq: ', 18)
('Joint states: ', [11.0, 12.0, 13.0, 14.0])
^Cstudent@robot-autonomy:~/sample/src/robot_arm_
('Seq: ', 1)
('Joint states: ', [1.0, 2.0, 3.0, 4.0])
('Seq: ', 2)
('Joint states: ', [7.0, 8.0, 9.0, 10.0])
('Seq: ', 3)
('Joint states: ', [11.0, 12.0, 13.0, 14.0])
^Cstudent@robot-autonomy:~/sample/src/robot_arm_
```

The robotic arm moving according to the various set of joint angles which are also printed on the screen

VIII. Testing

A physical and a ROS simulation test was done to confirm the proper communication of IMU with the microcontroller and the data published is as below:

For physical testing, the IMU was strapped to a SG90 servo motor which was provided an angle of rotation by the microcontroller and the angles recorded by the IMU at set PWM of the Arduino were recorded.



Analog Values (Accelerometer)		PWM value (p)
X	Y	
360-370	290-300	600
320-330	300-310	800
300-310	320-330	1100
290-300	350-360	1560
300-310	380-390	1800
320-330	400-410	2100
360-370	420-430	2400

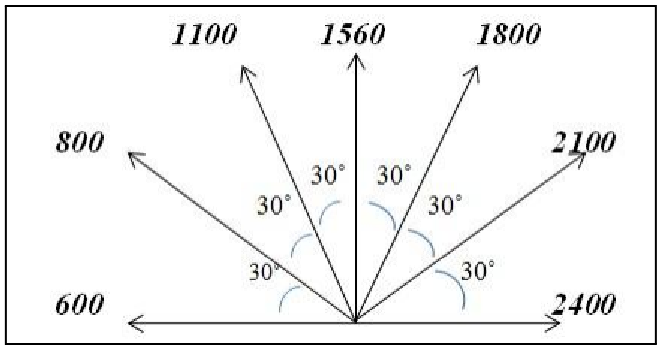


Fig. No. 12: (i) the setup for the recording test. The servo was held stationary against the table to nullify the z-axis accelerometer readings (ii) table listing ranges of X and Y values of accelerometer against the corresponding PWM value of the digital input of the Arduino (iii) representation of angles of rotation of the servo w.r.t. PWM values.

Conclusion

An open source resource platform with support for design guidelines, sensor fusion, filtering, kinematics, URDF parsing, and simulation is beneficial for any future developments of robotic arms with n degrees of freedom and makes the overall process pipeline much more easier for any such future developers. Its modular nature makes it available for a large number of applications in a wide range of fields. This project also promises to reduce the overall cost and developmental time of teams building any robotic systems. Application of kinematics and filtering also makes the code structure stable. ROS simulation adds credibility to the project and helps build confidence in achieving the hardware aspect of the same.

References

[1] J. J. Uicker, G. R. Pennock, and J. E. Shigley, 2003, Theory of Machines and Mechanisms, Oxford University Press, New York.

[2] J. M. McCarthy and G. S. Soh, 2010, Geometric Design of Linkages, Springer, New York.

[3] A. Aristidou, J. Lasenby, Y. Chrysanthou, A. Shamir. Inverse Kinematics Techniques in Computer Graphics: A Survey. Computer Graphics Forum, 37(6): 35-58, 2018.

[4] D. G. Luenberger. 1989. Linear and Nonlinear Programming. Addison Wesley.

[5] David Corinaldi, Luca Carbonari, and Massimo Callegari IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 3, NO. 2, APRIL 2018 735 Optimal Motion Planning for Fast Pointing Tasks With Spherical Parallel Manipulators

[6] Po-Wen Hsueh / Department of Mechanical Engineering, National Cheng Kung University The Introduction of precision positioning and compensation technology in GantryType platform

[7] Yisheng Guan, Kazuhito Yokoi, Olivier Strasse, and Abderrahmane Kheddar JRL, Intelligent Systems Research Institute National Institute of Advanced Industrial Science and Technology (AIST) On Robotic Trajectory Planning Using Polynomial Interpolations

[8] J. Brinker1, N. Funk1, P. Ingenlath1, Y. Takeda2, and B. Corves1Comparative Study of Serial-Parallel Delta Robots with Full Orientation Capabilities