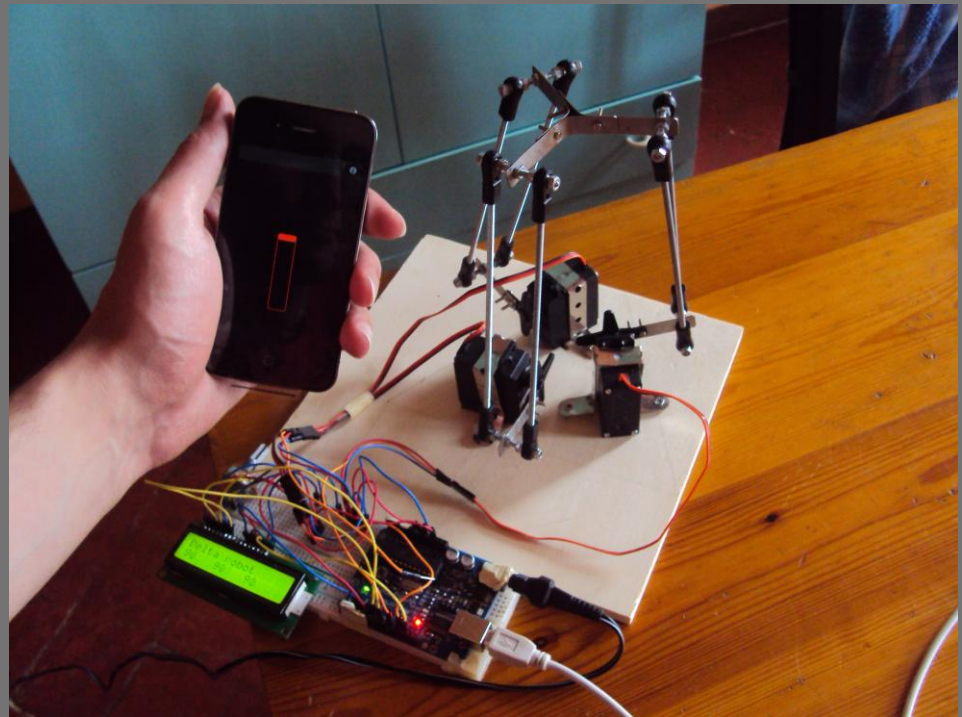


2010



Study of a parallel delta robot controlled via iPhone accelerometer

From kinematics theory to a practical implementation



Student: Filippo Sanfilippo, Professor: Domenico Prattichizzo
Università degli Studi di Siena, Faculty of Computer Engineering



Table of Contents

Introduction.....	4
Delta robot design	5
Applications	5
Problem definition.....	6
Kinematics	7
Inverse kinematics.....	7
Some key parameters.....	7
Equations	8
Forward Kinematics.....	10
Some key parameters.....	10
Equations	11
Sample build	13
My own delta robot.....	13
System architecture.....	13
Use cases	13
Architecture and behavior.....	14
Class diagram.....	14
Sequence diagram	15
How it works.....	15
Design choices	16
Position determination using accelerometers	16
Some implementation details	17
Processing server.....	17
Arduino client	20
Setting TouchOSC app	22
Electric schemes	24
Some consideration about forward kinematics implementation	25
Possible enhancements.....	26
Source code	27
Processing code (processing_server).	27
Arduino sketch (arduino_client).....	30
Table of figures	32

Bibliography.....	32
-------------------	----

To my dear parents and my whole family for their moral and economic support during my study.

Introduction

At the end of the robotics course taught by Professor Domenico Prattichizzo, I decided to carry out a little project to elaborate on the course topics. For this reason I decided to explore the field of industrial robots.

When one talks about industrial robots, most of people imagine robotic arms, or articulated robots, which paint, weld, move something, etc. But there is another interesting type of robot: the so-called parallel delta robot, which was invented in the early 80's in Switzerland by professor Reymond Clavel.

Below, the original technical drawing from U.S. Patent n. 4,976,582 is shown.

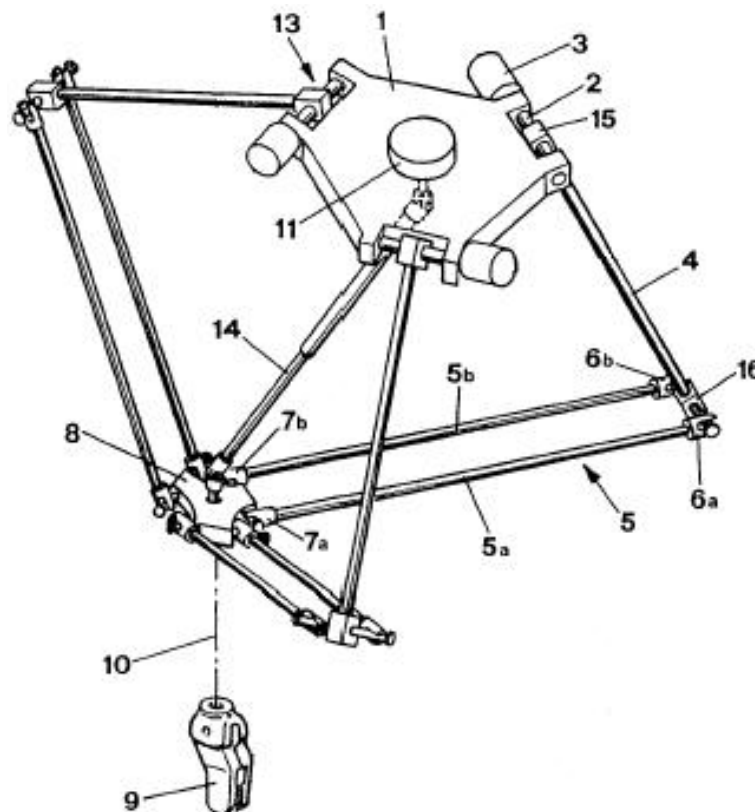


Figure 1 - technical drawing from U.S. Patent n. 4,976,582

Delta robot design

Before proceeding with the description of my work, I will briefly explain what a delta robot is and what its possible applications are.

A delta robot consists of three arms connected to universal joints at the base. The key design feature is the use of parallelograms in the arms, which maintains the orientation of the end effector (1). Delta robots are commonly used for grasping and packaging in factories because they can be quite fast.

Referring to figure 1, this kind of robot has two platforms: the upper platform (1) with three motors (n.3) mounted on it, and the lower platform (n.8) with an end effector (n.9). The platforms are connected by three arms that form three parallelograms – one for each side. The parallelograms restrain the orientation of the lower platform in order to keep it parallel to the working surface (table, conveyor belt and so on). The motors (n.3) set the position of the arms (n.4) and, thereby, the XYZ-position of the end effector, while the fourth motor (n.11) is used to rotate the end effector (2).

A delta robot has four degrees of freedom; three translational and one rotational.

However, the robot that I have built only has three degrees of freedom. In fact, it cannot turn about itself.

The greatest advantage of delta robots is speed. Typically, when a robot arm has to move not only a payload, but also all servos in each joint, the only moving part of the delta robot is its frame, which is usually made of lightweight composite materials.

Applications

Industries that take advantage of the Delta robot's high speed are the packaging, medical and pharmaceutical industries. Other possible applications include assembly tasks or operation in a clean room for electronic components. The Delta robot's structure can also be used to create haptic controllers.

The Delta robots available on the market typically operate in a cylindrical workspace that is 1 m in diameter and 0.2 m height.

Problem definition

I wanted to build my own delta robot, so I needed to solve two problems. Firstly, given the desired position of the end effector (for example, to catch a pancake in the point with coordinates x,y,z), it is necessary to determine the corresponding angles of each of three arms (joint angles) in order to set the motors (and, thereby, the end effector) in proper position for grasping. This process is known as inverse kinematics.

And, secondly, if the joint angles are known (for example, by reading the values of the motor encoders), it is necessary to determine the position of the end effector (i.e. to make some corrections regarding its current position). This is known as a forward kinematics problem.

To be more formal, let's look at the kinematic scheme of the delta robot. The platforms are two equilateral triangles: the fixed triangle with the motors is green, and the movable triangle with the end effector is pink. The joint angles are θ_1 , θ_2 and θ_3 , and the point E_0 is the end effector's position with coordinates (x_0, y_0, z_0) .

To solve inverse kinematics problems we have to create a function with the coordinates (x_0, y_0, z_0) of E_0 as parameters, which returns $(\theta_1, \theta_2, \theta_3)$. A forward kinematics function uses $(\theta_1, \theta_2, \theta_3)$ to return (x_0, y_0, z_0) .

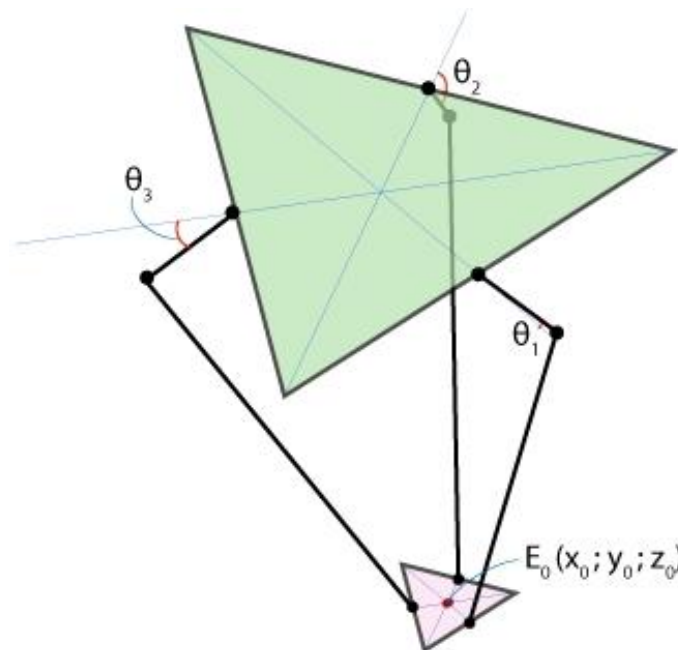


Figure 2 - Kinematic scheme of the delta robot

Kinematics

Inverse kinematics

Some key parameters

First, let's determine some key parameters of our robot's geometry. Let's designate the side of the fixed triangle as f , the side of the end effector triangle as e , the length of the upper joint as r_f , and the length of the parallelogram joint as r_e . These are physical parameters that are determined by the robot. The reference frame will be chosen with the origin at the fixed triangle's centre of symmetry, as shown below, so that the z-coordinate of the end effector will always be negative.

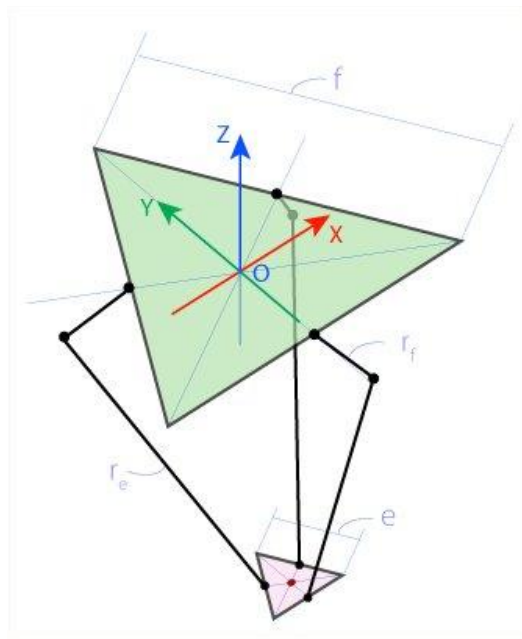


Figure 3 - some key parameters of our robot's geometry

Because of the robot's design, joint $\overline{F_1 J_1}$ can only rotate in the YZ plane (see fig. below), forming a circle centered in point F_1 with radius r_f . Unlike F_1, J_1 and E_1 are so-called universal joints, which means that $\overline{E_1 J_1}$ can rotate freely relatively to E_1 , forming a sphere centered in point E_1 with radius r_e .

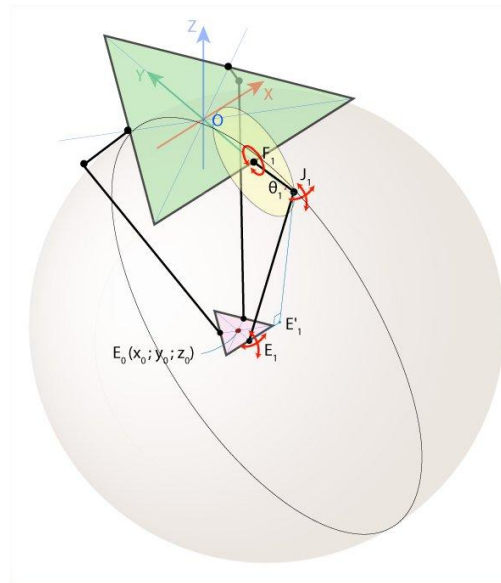


Figure 4 – how to find point J1

The intersection of this sphere and the YZ plane is a circle centered in point E'_1 with radius $\overline{E'_1J_1}$, where E'_1 is the projection of the point E_1 on the YZ plane. The point J_1 can be found now as the intersection of two circles of known radius centered in E'_1 and F_1 respectively (we should choose only one intersection point with a smaller Y-coordinate). And if we know J_1 , we can calculate the angle of θ_1 .

Such algebraic simplicity follows from our good choice regarding the frame of reference, which causes joint $\overline{F_1J_1}$ to move in the YZ plane only. As such, we can completely omit the X coordinate.

Equations

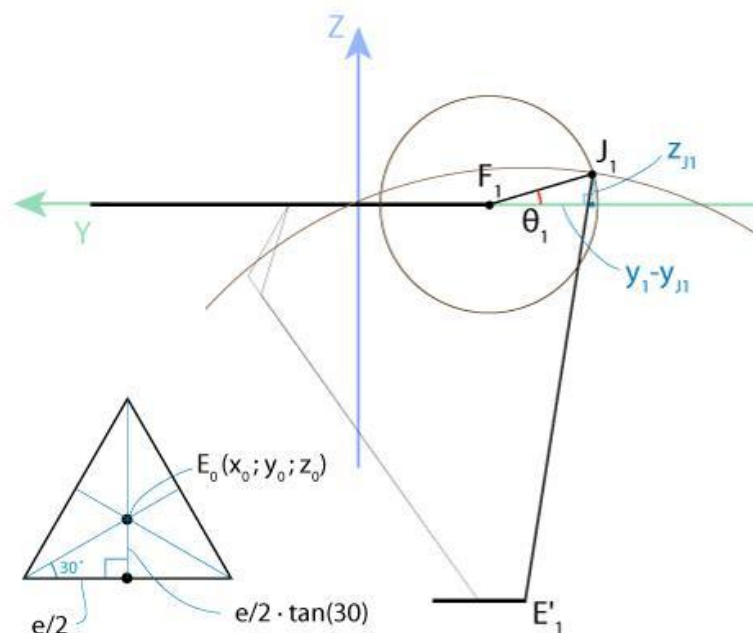


Figure 5 – how to find theta1

$$E_0 = (x_0, y_0, z_0)$$

$$\overline{EE_1} = \frac{e}{2} \tan 30^\circ = \frac{e}{2\sqrt{3}}$$

$$E_1 = \left(x_0, y_0 - \frac{e}{2\sqrt{3}}, z_0\right) \Rightarrow E'_1 = \left(0, y_0 - \frac{e}{2\sqrt{3}}, z_0\right)$$

$$\overline{E_1E'_1} = x_0 \Rightarrow \overline{E'_1J_1} = \sqrt{E_1J_1^2 - \overline{E_1E'_1}^2} = \sqrt{r_e^2 - x_0^2}$$

$$F_1 = \left(0, -\frac{f}{2\sqrt{3}}, 0\right)$$

$$\begin{cases} (y_{J_1} - y_{F_1})^2 + (z_{J_1} - z_{F_1})^2 = r_f^2 \\ (y_{J_1} - y_{E'_1})^2 + (z_{J_1} - z_{E'_1})^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow \begin{cases} \left(y_{J_1} - \frac{f}{2\sqrt{3}}\right)^2 + z_{F_1}^2 = r_f^2 \\ \left(y_{J_1} - y_0 + \frac{e}{2\sqrt{3}}\right)^2 + (z_{J_1} - z_0)^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow$$

$$\Rightarrow J_1 = (0, y_{J_1}, z_{J_1})$$

$$\theta_1 = \tan^{-1}\left(\frac{z_{J_1}}{y_{F_1} - y_{J_1}}\right)$$

For the remaining angles theta2 and theta3, we should take advantage of the delta robot's symmetry. First, let's rotate the XY plane 120 degrees counterclockwise around the Z-axis, as it is shown below.

We've got a new reference frame X'Y'Z', and with this frame we can find the angle theta2 using the same algorithm that we used to find theta1. The only change is that we need to determine new coordinates x'_0 and y'_0 for the point E_0 , which can be easily done using the corresponding rotation matrix. To find the angle theta3 we have to rotate the frame of reference clockwise, starting from the initial position.

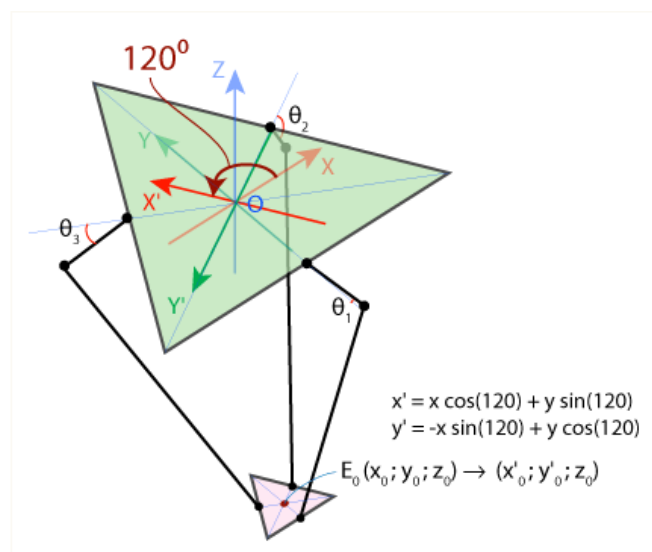


Figure 6 – how to find theta2 and theta3

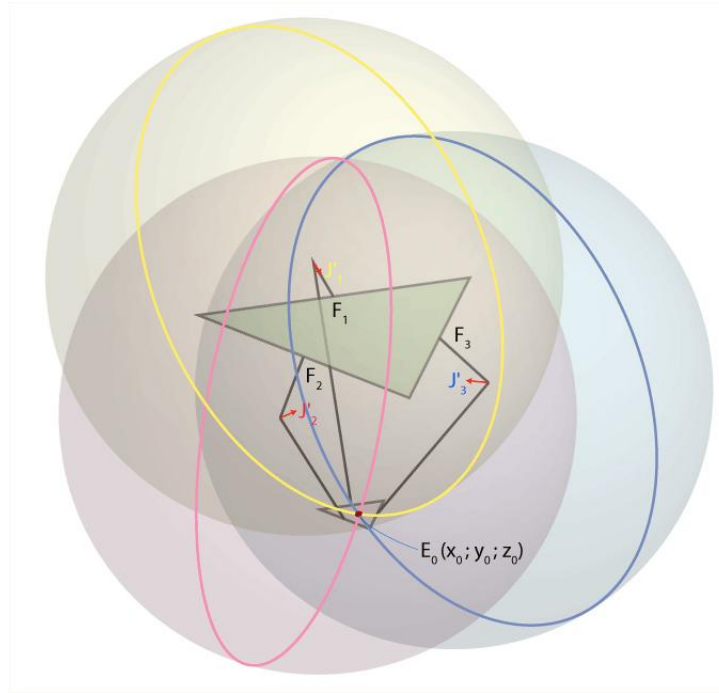


Figure 8 - intersection of three spheres

Equations

So, to find the coordinates (x_0, y_0, z_0) of the point E_0 , we need to solve a set of three equations like

$$(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2 = r_e^2$$

where the coordinates of sphere centers (x_j, y_j, z_j) and the radius r_e are known. First, let's find the coordinates of points J'_1, J'_2, J'_3 :

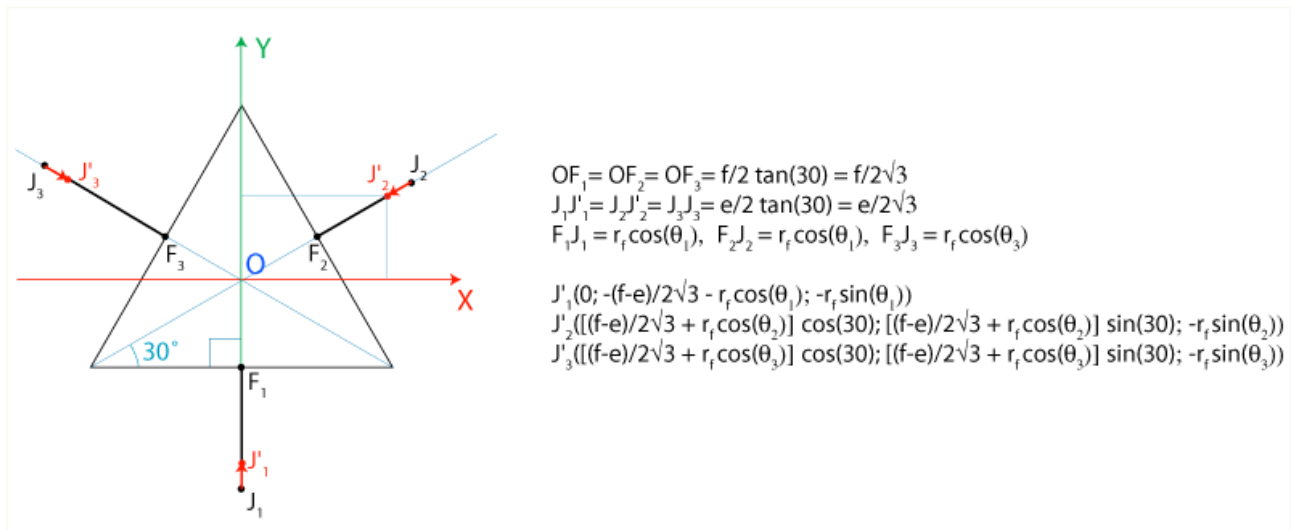


Figure 9 - how to find J1', J2', J3' coordinates

In the following equations I'll designate the coordinates of points J_1, J_2, J_3 as $(x_1, y_1, z_1), (x_2, y_2, z_2)$ and (x_3, y_3, z_3) . Please note that $x_0 = 0$. Here are the equations of the three spheres:

$$\begin{cases} x^2 + (y - y_1)^2 + (z - z_1)^2 = r_e^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_e^2 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r_e^2 \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} x^2 + y^2 + z^2 - 2y_1y - 2z_1z = r_e^2 - y_1^2 - z_1^2 & (1) \\ x^2 + y^2 + z^2 - 2x_2x - 2y_2y - 2z_2z = r_e^2 - x_2^2 - y_2^2 - z_2^2 & (2) \\ x^2 + y^2 + z^2 - 2x_3x - 2y_3y - 2z_3z = r_e^2 - x_3^2 - y_3^2 - z_3^2 & (3) \end{cases}$$

$$w_i = x_i^2 + y_i^2 + z_i^2$$

$$\begin{cases} x_2x + (y_1 - y_2)y + (z_1 - z_2)z = \frac{w_1 - w_2}{2} & (4) = (1) - (2) \\ x_3x + (y_1 - y_3)y + (z_1 - z_3)z = \frac{w_1 - w_3}{2} & (5) = (1) - (3) \\ (x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = \frac{w_1 - w_3}{2} & (6) = (2) - (3) \end{cases}$$

From (4)-(5):

$$x = a_1z + b_1 \quad (7)$$

$$y = a_2z + b_2 \quad (8)$$

$$a_1 = \frac{1}{d} [(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)]$$

$$a_2 = \frac{1}{d} [(z_2 - z_1)x_3 - (z_3 - z_1)x_2]$$

$$d = (y_2 - y_1)x_3 - (y_3 - y_1)x_2$$

Now we can substitute (7) and (8) in (1):

$$(a_1^2 + a_2^2 + 1)z^2 + 2[a_1 + a_2(b_2 - y_1) - z_1]z + [b_1^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2] = 0$$

And finally, we need to solve this quadric equation and find z_0 (we should choose the smallest negative root), and then calculate x_0 and y_0 from eq. (7) and (8).

Sample build

My own delta robot

Some weeks ago I saw a video and a tutorial (3) of a little delta robot and I was intrigued by the complexity of the movements that it could make, despite its simple structure.

So I thought, I am going to make one.

Also, I thought it would be interesting to be able to control it using an iPhone, taking advantage of its accelerometer.

So I bought three servos, some ball joints and some aluminum strips. I already had the wooden base laying around, as well as some nuts and bolts.

I decided to use an Arduino microcontroller to manage the robot via USB.

I decided to program the Arduino to read OSC serial messages, and I programmed an interface for the iPhone that reads the accelerometer's data and sends them to a server application via OSC messages. When the server receives them, it calculates inverse kinematics and sends the joint angles' values to the Arduino. The angles are used to drive the motors.

System architecture

Use cases

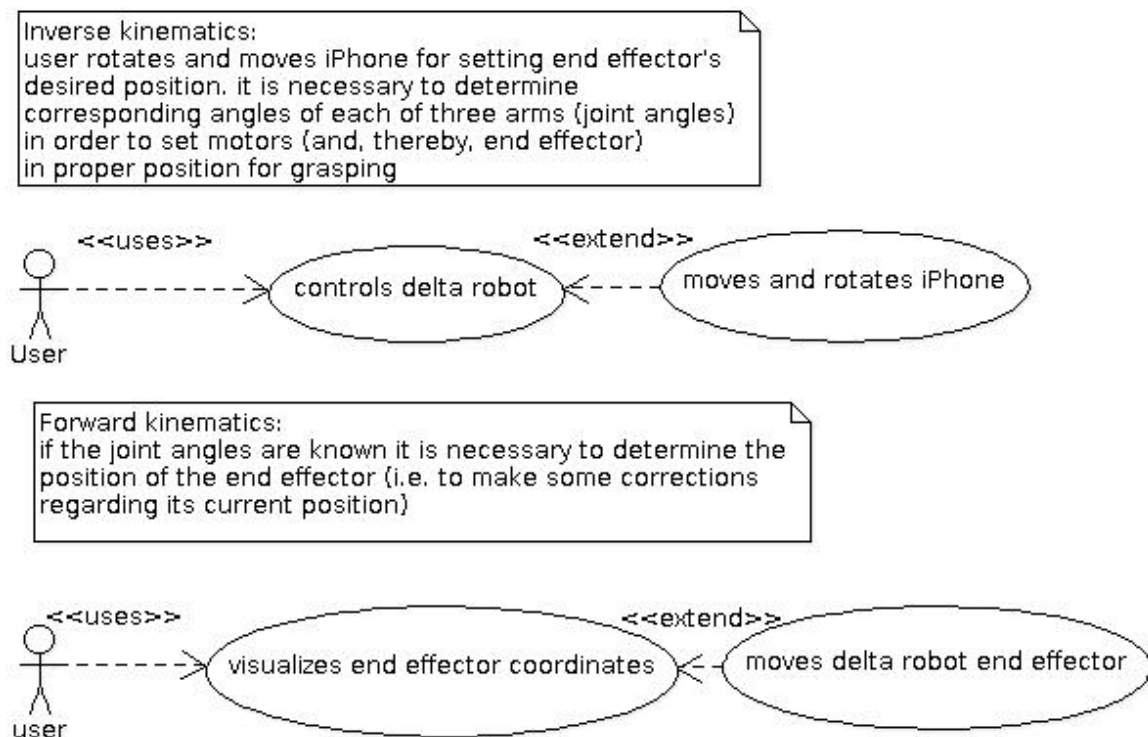


Figure 10 - use cases

Architecture and behavior

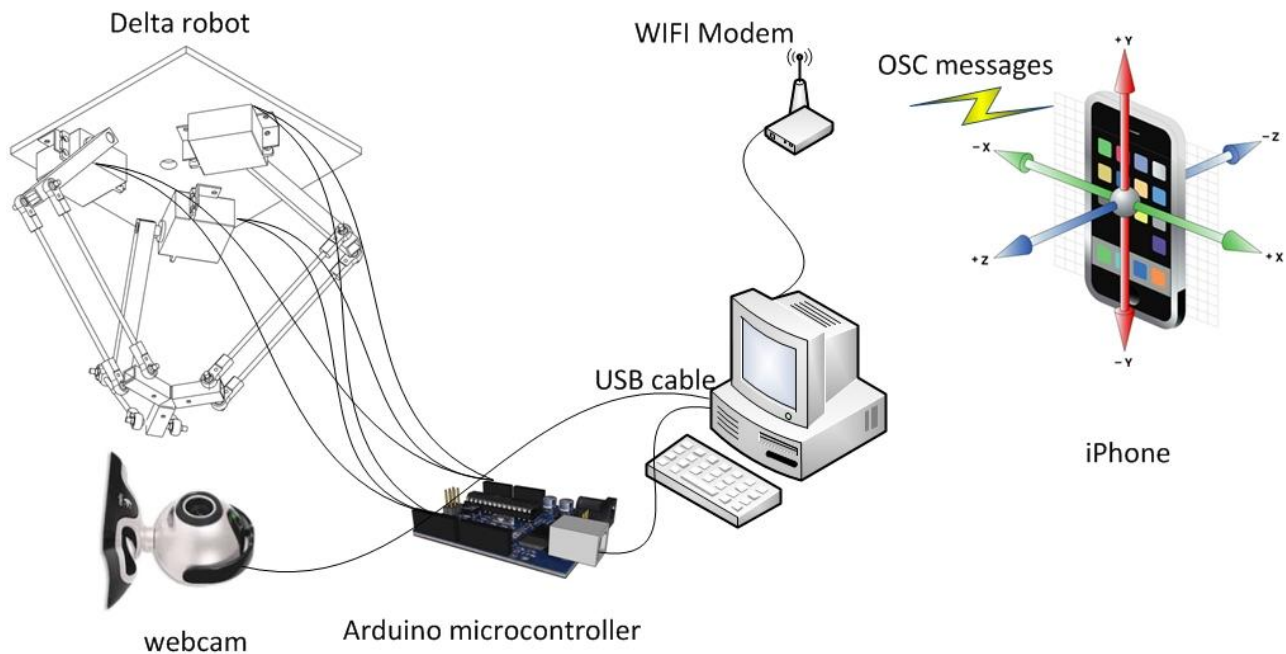


Figure 11 - system architecture

Class diagram

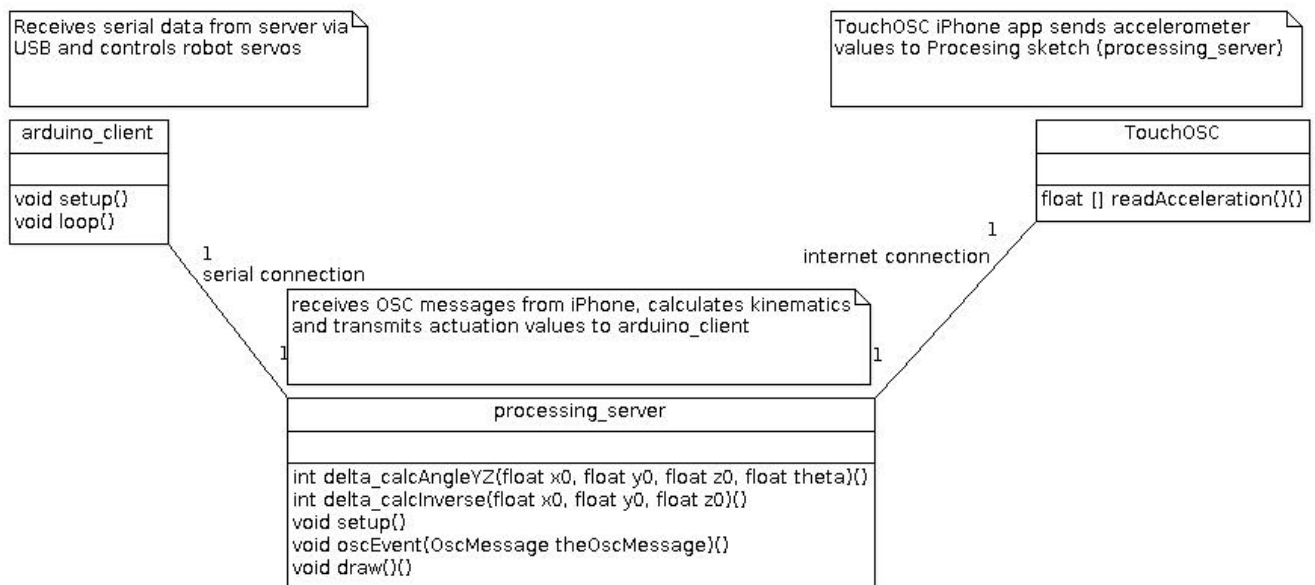


Figure 12 - class diagram

Sequence diagram

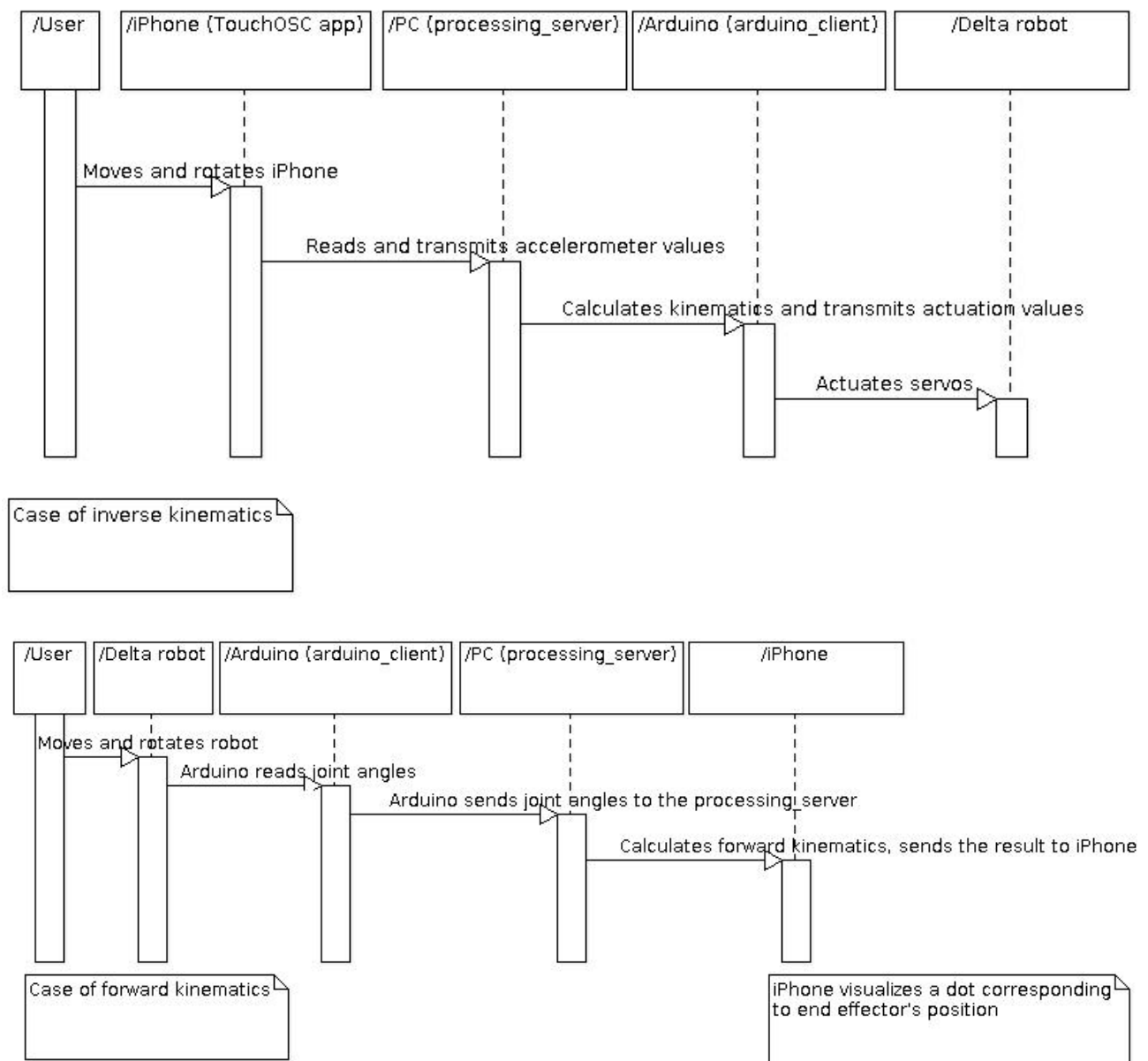


Figure 13 - sequence diagrams

How it works

I decided to control my delta robot with an iPhone by using its accelerometer. As such, I needed to create a communication channel between the iPhone and the Arduino microcontroller. To do this, I decided to use OSC messages.

OSC stands for Open Sound Control. It is an open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology.

OSC messages are commonly transported across the internet and within home and studio subnets using (UDP/IP, Ethernet). OSC messages between gestural controllers are usually transmitted over serial endpoints of USB by being wrapped in the SLIP protocol. This technology is exactly what I needed to transmit data from the iPhone accelerometer to the Arduino that controls the servos via USB.

Design choices

To program the Arduino and design the server application I used the available free and flexible tools.

Regarding the iPhone I decided to use a universal iPhone / iPod Touch / iPad application, called TouchOSC, that allows for sending and receiving Open Sound Control messages over a Wi-Fi network using the UDP protocol.

TouchOSC supports full multi-touch operation. Five controls can be used at the same time. Additionally, the program can send Accelerometer data, so it is perfect for my needs. The application comes with five default layouts that are organized in multiple pages but custom layouts can be constructed using the TouchOSC Editor application (4).

The TouchOSC iPhone app communicates with the Processing sketch (processing_server class) over a wifi network, which in turn sends serial data to the Arduino board. The latter controls the robot servos.

Position determination using accelerometers

The signal obtained from iPhone accelerometer has to be processed as there is no direct conversion between acceleration and position. In order to obtain position a double integral must be applied to the signal.

Ideally the position (x) of a particle at any time (t) can be determined from the time dependant acceleration of that particle. In the following equation the integration constants have been explicitly included:

$$x(t) = \int_0^t \int_0^t a(t) dt dt + \int_0^t v_0 dt + x_0$$

where:

- a is the acceleration of the object;
- v_0 is the initial velocity of the object;
- x_0 is the initial position of the object.

The acceleration term can be measured directly using the iPhone accelerometer. However, there are associated errors in these sensor measurements. Care must be taken in analyzing the impact of these sensor errors.

A simplified model of the output of an accelerometer (measured voltage) is as follows. (5)

$$V_m(t) = S(1 + \delta S)a(t) + V_b + \delta V_b$$

where:

- V_m is the measured voltage;
- S is the sensitivity;
- δS is the sensitivity error (including temperature, ratiometric, cross axis errors);
- V_b is the voltage bias;
- δV_b is the voltage bias error (including temperature, ratiometric, noise errors);

- a is the acceleration to be measured.

Writing this in terms of an apparent measured acceleration we have:

$$a_m(t) = \frac{V_m(t) - V_b}{S} = a(t) + \delta S a(t) + \frac{\delta V_b}{S}$$

Assuming that the initial position and velocity are zero ($x_0 = 0, v_0 = 0$), substituting the last equation into the first one and integrating twice to obtain the position as a function of time yields:

$$x(t) = \int_0^t \int_0^t a_m(t) dt dt = \int_0^t \int_0^t a(t) dt dt + \delta S \int_0^t \int_0^t a(t) dt dt + \frac{\delta V_b}{S} + \int_0^t \int_0^t dt dt$$

The first term is the position data that I want to determine.

The second and third terms are the integration of the intrinsic sensor errors. The error from these terms can become egregious very quickly since they increase quadratically with time.

It is clear that care must be taken when attempting the above double integration. In many applications this short-term drift renders the double integration useless without an independent position check. In the medium term (many hours) the errors can be bounded through the use of filters and control loops. However, the errors are still large without very high precision sensors or secondary position checks/corrections.

Some implementation details

Processing server

When the processing server is launched, it starts the oscP5 protocol, listening for incoming messages at port 12000.

Moreover, I decided to put a camera below the robot. In this way it is possible to display the exactly position of the robot's end effector. So, inside the setup method, there is a line that initialize the camera using GS library for processing.

```
void setup() {  
  size(640, 480);  
  
  myPort = new Serial(this, Serial.list()[0], 115200);  
  oscP5 = new OscP5(this, 12000);  
  
  cam = new GSCapture(this, 640, 480);  
  
  frameRate(100);  
  noCursor();  
}
```

When the user rotates the iPhone, the TouchOSC application sends an OSC message to the processing server.

So, how I said before, it is necessary to transform the acceleration values received into position coordinates. In my code, I decided that the iPhone device would start with an approximate velocity and position of zero (this is how I initialized my values). The proper way to proceed would be as follows:

- Extrapolate the gravity vector from the acceleration data using a low pass filter.
- subtract the normalized gravity vector from each x, y, and z, and get the movement's acceleration.
- then, integrate this acceleration value with respect to time, to get the velocity.
- Finally, integrate this velocity again with respect to time, and find the position.

Once the server obtains the iPhone orientation, it can apply the kinematic functions and send the actuation values to the Arduino via USB.

Wanting to prioritize the rate of project development, I decided to use the tilt of the iPhone as spatial coordinates. In the future it might be possible to follow the correct procedure simply by adding the portion of code that transforms the accelerometer data into displacement coordinates.

```
void oscEvent(OscMessage theOscMessage) {
  if(theOscMessage.checkAddrPattern("/accxyz")==true) {
    xrot_targ = (theOscMessage.get(0).floatValue()*90);
    zrot_targ = (theOscMessage.get(1).floatValue()*90)*-1;
    orientation = theOscMessage.get(2).floatValue();
  }
  if(theOscMessage.checkAddrPattern("/1/fader1")==true) {
    yAmplify = theOscMessage.get(0).floatValue(); // get the second osc argument
    println(" values: "+yAmplify);
  }
}
```

To calculate the kinematics, first I define some variables:

```
// robot geometry
float e = 70.0;      // side of end effector triangle
float f = 170.0;     // side of fixed triangle
float re = 150.0;    // length of parallelogram joint
float rf = 40.0;     // length of upper joint

// trigonometric constants
float sqrt3 = sqrt(3.0);
float pi = 3.141592653; // PI
float sin120 = sqrt3/2.0;
float cos120 = -0.5;
float tan60 = sqrt3;
float sin30 = 0.5;
float tan30 = 1/sqrt3;
```

To calculate the inverse kinematics I implemented a helper function:

```
// inverse kinematics
// helper function, calculates angle theta1 (for YZ-plane)
int aux = 0;
int delta_calcAngleYZ(float x0, float y0, float z0) {
  float theta = 0;
  float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30
```

```
y0 -= 0.5 * 0.57735 * e; // shift center to edge
// z = a + b*y
float a = (x0*x0 + y0*y0 + z0*z0 + rf*rf - re*re - y1*y1)/(2*z0);
float b = (y1-y0)/z0;
// discriminant
float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
if (d < 0) return -1; // non-existing point
float yj = (y1 - a*b - sqrt(d))/(b*b + 1); // choosing outer point
float zj = a + b*yj;
theta = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
if(aux==1)
    t1=theta;
if(aux==2)
    t2=theta;
if(aux==3)
    t3=theta;

return 0;
}
```

The inverse kinematics:

```
// inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)
// returned status: 0=OK, -1=non-existing position
int delta_calcInverse(float x0, float y0, float z0) {

    aux=1;
    int status = delta_calcAngleYZ(x0, y0, z0);
    aux=2;
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 + y0*sin120,
y0*cos120-x0*sin120, z0); // rotate coords to +120 deg
    aux=3;
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 - y0*sin120,
y0*cos120+x0*sin120, z0); // rotate coords to -120 deg
    return status;
}
```

I also implemented a method that draws the base triangle and the robot's end effector.

Moreover, it invokes the function that calculates inverse kinematics and sends the results to the microcontroller through the serial connection.

```
void draw() {
    if (cam.available() == true) {
        cam.read();
        image(cam, 0, 0);
        // The following does the same, and is faster when just drawing the image
        // without any additional resizing, transformations, or tint.
        //set(160, 100, cam);
    }
    //background(0);
    stroke(255);
    //line(width/2, height, 0, 80);
    //line(width/2, height, width, 80);
    line(0, height-80, width, height-80);
    line(0, height-80, width/2, 0);
    line(width, height-80, width/2, 0);

    // Basic value smoothing
}
```

```
if (xrot_targ > xrot) {
    xrot = xrot + ((xrot_targ - xrot) / dampSpeed);
} else {
    xrot = xrot - ((xrot - xrot_targ) / dampSpeed);
}
println("xrot "+xrot);

if (zrot_targ > zrot) {
    zrot = zrot + ((zrot_targ - zrot) / dampSpeed);
} else {
    zrot = zrot - ((zrot - zrot_targ) / dampSpeed);
}

println("zrot "+zrot);

println("orientation "+orientation);

result=delta_calcInverse(xAccelerometerAmplify*xrot,
zAccelerometerAmplify*zrot, -yAmplify*13-160);

println(+result);
println("t1 "+t1);
println("t2 "+t2);
println("t3 "+t3);

strokeWeight(3);

line(width/2,400,xAccelerometerAmplify*xrot+width/2,zAccelerometerAmplify*zrot+height/2);
line(190,height/2-
80,xAccelerometerAmplify*xrot+width/2,zAccelerometerAmplify*zrot+height/2);
line(450,height/2-
80,xAccelerometerAmplify*xrot+width/2,zAccelerometerAmplify*zrot+height/2);

if(result ==0){
    if (abs(oldT1 - t1) > threshold || abs(oldT2 - t2) > threshold || abs(oldT3
- t3) > threshold) {
        myPort.write(253);
        myPort.write((int) (t1+90.0-14.0));
        oldT1 = t1;
        myPort.write(254);
        myPort.write((int) (t2+90.0-10.0));
        oldT2 = t2;
        myPort.write(255);
        myPort.write((int) (t3+90.0-00.0));
        oldT3 = t3;
    }
}
```

Arduino client

When the application is launched it selects the pin mode, initializes the servos and opens the channel for serial communication. I also provided to connect an LCD monitor to the Arduino microcontroller in order to display the joints angles in real time. So there are some line of my code that provide to manage the LCD display.

```
void setup() {  
  // set up the LCD's number of rows and columns:  
  lcd.begin(16, 2);  
  lcd.print("Delta robot");  
  
  pinMode(13, OUTPUT);  
  
  servo1.attach(10); // attaches the servo on pin 10 to the servo object  
  servo2.attach(11); // attaches the servo on pin 11 to the servo object  
  servo3.attach(12); // attaches the servo on pin 12 to the servo object  
  //pinmode  
  Serial.begin(115200);  
}
```

Inside the loop, the application monitors the serial channel and if any data is available it reads the values and commands the servos.

```
void loop() {  
  if (Serial.available() > 0) {  
    // read the incoming byte:  
    data = Serial.read();  
  
    if (data == 255) {  
      delay(7);  
      if (Serial.available() > 0) {  
        servo1Pos = Serial.read();  
        if (servo1Pos < 30)  
          servo1Pos = 30;  
        servo1.write(servo1Pos);  
      }  
    }  
  
    if (data == 254) {  
      delay(7);  
      if (Serial.available() > 0) {  
        servo2Pos = Serial.read();  
        if (servo2Pos < 30)  
          servo2Pos = 30;  
        servo2.write(servo2Pos);  
      }  
    }  
  
    if (data == 253) {  
      delay(7);  
      if (Serial.available() > 0) {  
        servo3Pos = Serial.read();  
        if (servo3Pos < 30)  
          servo3Pos = 30;  
        servo3.write(servo3Pos);  
      }  
    }  
  }  
  lcd.setCursor(0, 1);  
  itoa(servo3Pos, str, 10);  
  lcd.print(str);  
  lcd.setCursor(5, 1);  
  itoa(servo2Pos, str, 10);  
  lcd.print(str);  
  lcd.setCursor(10, 1);
```

```
itoa(servo1Pos, str, 10);  
lcd.print(str);  
  
SoftwareServo::refresh();  
}
```

Setting TouchOSC app

Regarding the iPhone, the TouchOSC application must be installed. It is available on the App Store. The most important TouchOSC function used is the ability to read the accelerometer's data.

Also, I decided to use a slider to control the robot's end effector position along the z axis. So, it is necessary to create a proper layout using the TouchOSC Editor. A slider has to be added as shown below.

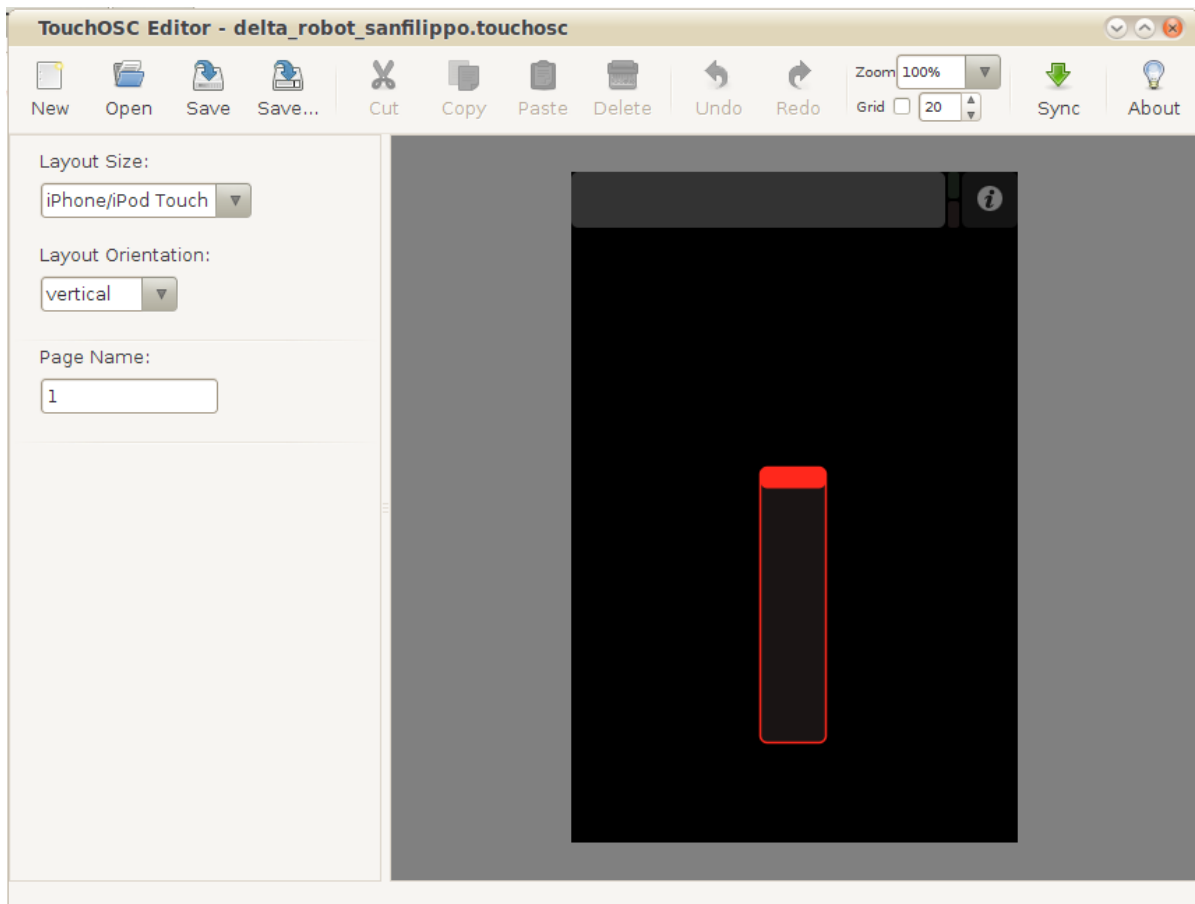


Figure 14 - TouchOSC Editor

This custom layout has to be added to the TouchOSC app's layouts by clicking on synchronize.

Moreover, the accelerometer has to be enabled from the menu options as shown in the next page.



Figure 15 - TouchOSC options

In addition, the server's IP address and outgoing/incoming ports must be set. 1200 has to be chosen as the outgoing port (in accordance with the processing server).

Electric schemes

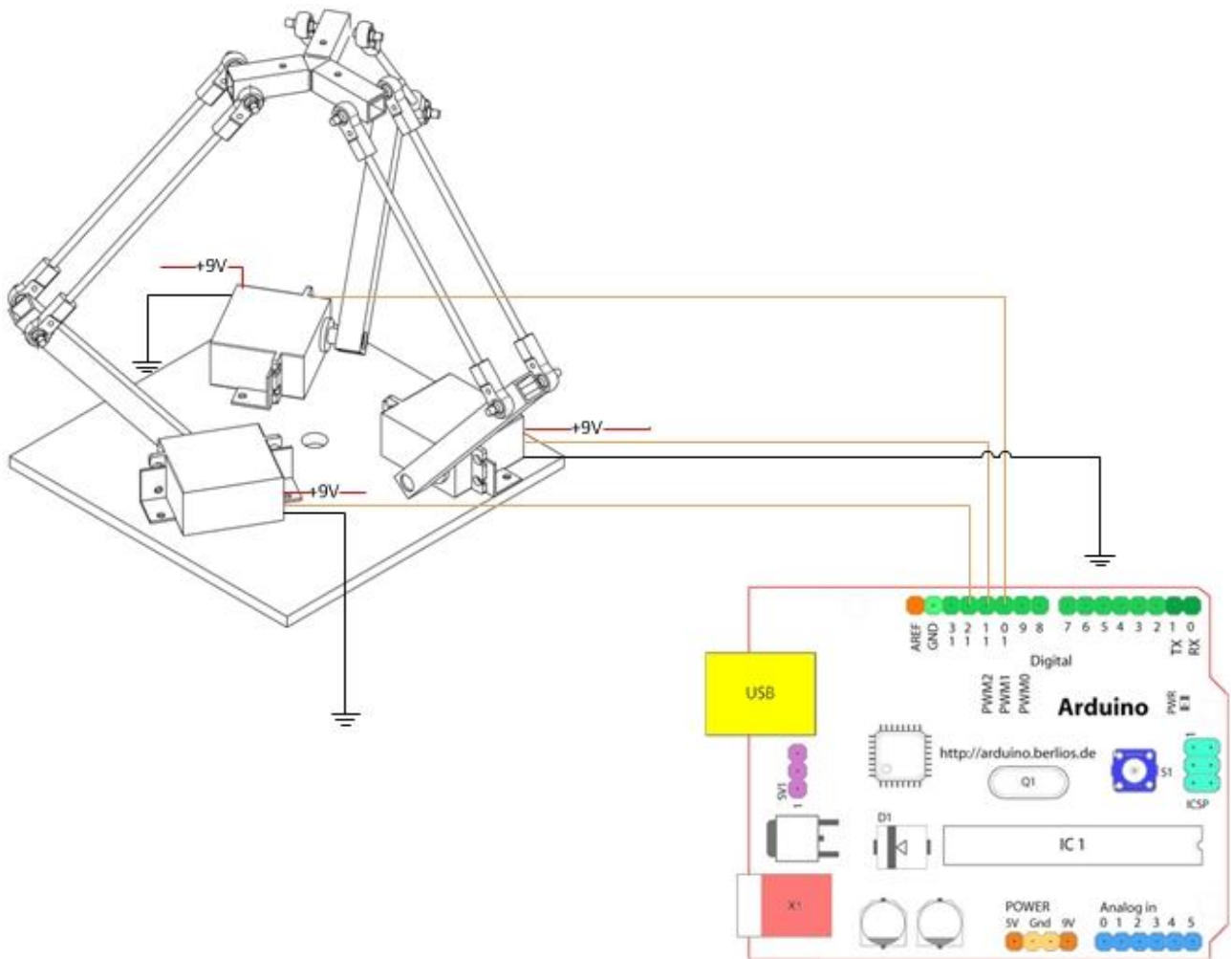


Figure 16 – servo motors interconnection

I also connected an LCD monitor to the Arduino microcontroller to display the joints angles in real time. Below the electric scheme.

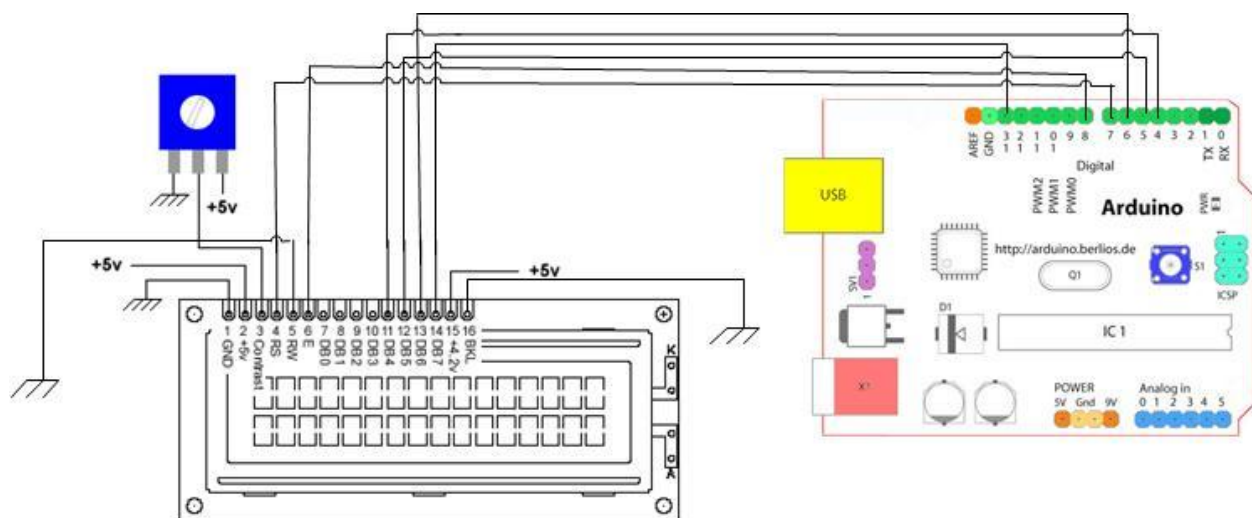


Figure 17 - LCD display interconnection

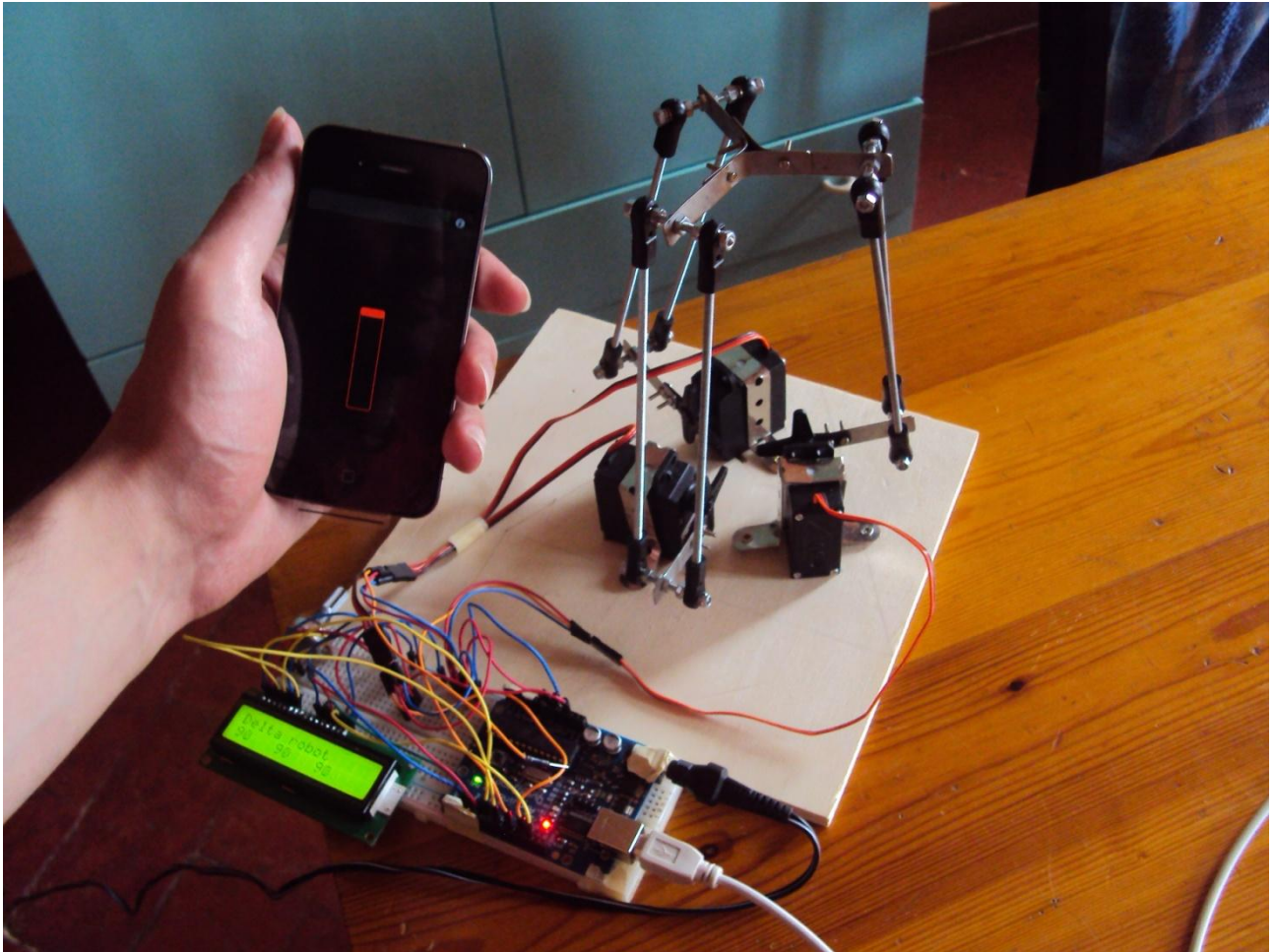


Figure 18 - my delta robot

Some consideration about forward kinematics implementation

At present, I have not yet implemented forward kinematics because there is no servo-to-controller communication. It is one-way only. Servo-to-controller communication would only be possible by using other external components, for instance, a pot coupled to the servo wheel, or perhaps tapping into the servo's internal position pot.

Another option I might want to look into is the "robotic servo".

A robotic servo is a motor that has an integrated microcontroller, and two-way communication is permitted using a serial protocol (I2C or SPI, or something similar); typically, they can be daisy chained on a "bus" so that a large number of servos can be connected and controlled using a minimum of data lines from the controller (in this case, the Arduino). They generally include direct feedback reading capabilities for the internal components (sometimes even temperature and other variables).

They were designed for use in walking robots (usually hexapods and bipedal robots), where the need to have feedback from a large number of servos was paramount. They also tend to be fairly robust and powerful (generally using ball bearings for the shafts and metal gear trains). As such, they are generally much more expensive than a typical servo.

While I could modify a servo to add a fourth line in order to monitor the potentiometer directly. On some servos this can be difficult, and I have read some reports of the extra line interfering with the servo's own feedback circuitry, causing problems with positioning (as well as jittery output on the added line). But it is definitely worth trying, especially if we are trying to keep costs low.

Possible enhancements

Some improvements that could be made to the project are:

- implementation of the correct procedure for detecting motion from the iPhone's accelerometer data
- implementation of forward kinematics;
- standalone, all calculations done on the Arduino;
- implementation of a specific iPhone / iPod / iPad application which lets us receive streaming video from the webcam. Moreover, it could be very interesting to use augmented reality for visualizing on iPhone's display the robot's end effector position and also virtual objects to grasp;
- implementation of vision algorithms that can detect objects to grasp.

Source code

Processing code (processing_server).

Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Open up the Processing Development Environment application and type the below code into the editor. Compile the sketch and run the code.

```
import processing.serial.*;
import oscP5.*;
import codeanticode.gsvideo.*;

Serial myPort;      // The serial port:
OscP5 oscP5;
GSCapture cam;

float t1;  //servo angle t for 'theta', 1 for servo 1
float t2;
float t3;
float oldT1;
float oldT2;
float oldT3;

int result=0;

float threshold = 3.0;

float xAccelerometerAmplify = 0.82;
float zAccelerometerAmplify = 0.95;

float xrot = 0;
float zrot = 0;

float xrot_targ = 0;
float zrot_targ = 0;
float orientation = -170;

float dampSpeed = 5;

// CODE FROM http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/
// robot geometry
float e = 70.0;      // the side of end effector triangle
float f = 170.0;     // the side of the fixed triangle
float re = 150.0;    // the length of the parallelogram joint
float rf = 40.0;     // the length of the upper joint

// trigonometric constants
float sqrt3 = sqrt(3.0);
float pi = 3.141592653;  // PI
float sin120 = sqrt3/2.0;
float cos120 = -0.5;
float tan60 = sqrt3;
float sin30 = 0.5;
float tan30 = 1/sqrt3;

// inverse kinematics
// helper functions, calculates angle theta1 (for YZ-plane)
int stupid = 0;
```

```
int delta_calcAngleYZ(float x0, float y0, float z0) {
    float theta = 0;
    float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30
    y0 -= 0.5 * 0.57735 * e; // shift center to edge
    // z = a + b*y
    float a = (x0*x0 + y0*y0 + z0*z0 + rf*rf - re*re - y1*y1)/(2*z0);
    float b = (y1-y0)/z0;
    // discriminant
    float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
    if (d < 0) return -1; // non-existing point
    float yj = (y1 - a*b - sqrt(d))/(b*b + 1); // choosing outer point
    float zj = a + b*yj;
    theta = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
    if(stupid==1)
        t1=theta;
    if(stupid==2)
        t2=theta;
    if(stupid==3)
        t3=theta;

    return 0;
}

// inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)
// returned status: 0=OK, -1=non-existing position
int delta_calcInverse(float x0, float y0, float z0) {

    stupid=1;
    int status = delta_calcAngleYZ(x0, y0, z0);
    stupid=2;
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 + y0*sin120,
y0*cos120-x0*sin120, z0); // rotate coords to +120 deg
    stupid=3;
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 - y0*sin120,
y0*cos120+x0*sin120, z0); // rotate coords to -120 deg
    return status;
}

void setup() {
    size(640,480);

    myPort = new Serial(this, Serial.list()[0], 115200);
    oscP5 = new OscP5(this,12000);

    cam = new GSCapture(this, 640, 480);

    frameRate(100);
    noCursor();
}

void draw() {
    if (cam.available() == true) {
        cam.read();
        image(cam, 0, 0);
        // The following does the same, and is faster when just drawing the image
        // without any additional resizing, transformations, or tint.
        //set(160, 100, cam);
    }
}
```

```
}
//background(0);
stroke(255);
//line(width/2, height, 0, 80);
//line(width/2, height, width, 80);
line(0, height-80, width, height-80);
line(0, height-80, width/2, 0);
line(width, height-80, width/2, 0);

// Basic value smoothing

if (xrot_targ > xrot) {
  xrot = xrot + ((xrot_targ - xrot) / dampSpeed);
} else {
  xrot = xrot - ((xrot - xrot_targ) / dampSpeed);
}
println("xrot "+xrot);

if (zrot_targ > zrot) {
  zrot = zrot + ((zrot_targ - zrot) / dampSpeed);
} else {
  zrot = zrot - ((zrot - zrot_targ) / dampSpeed);
}

println("zrot "+zrot);

println("orientation "+orientation);

result=delta_calcInverse(xAccelerometerAmplify*xrot,
zAccelerometerAmplify*zrot, -yAmplify*13-160);

println(+result);
println("t1 "+t1);
println("t2 "+t2);
println("t3 "+t3);

strokeWeight(3);

line(width/2,400,xAccelerometerAmplify*xrot+width/2,zAccelerometerAmplify*zrot+height/2);
  line(190,height/2-
80,xAccelerometerAmplify*xrot+width/2,zAccelerometerAmplify*zrot+height/2);
  line(450,height/2-
80,xAccelerometerAmplify*xrot+width/2,zAccelerometerAmplify*zrot+height/2);

if(result ==0){
  if (abs(oldT1 - t1) > threshold || abs(oldT2 - t2) > threshold || abs(oldT3
- t3) > threshold) {
    myPort.write(253);
    myPort.write((int) (t1+90.0-14.0));
    oldT1 = t1;
    myPort.write(254);
    myPort.write((int) (t2+90.0-10.0));
    oldT2 = t2;
    myPort.write(255);
    myPort.write((int) (t3+90.0-00.0));
```

```
        oldT3 = t3;
    }
}

float yAmplify;
void oscEvent(OscMessage theOscMessage) {
    if(theOscMessage.checkAddrPattern("/accxyz")==true) {
        xrot_targ = (theOscMessage.get(0).floatValue()*90);
        zrot_targ = (theOscMessage.get(1).floatValue()*90)*-1;
        orientation = theOscMessage.get(2).floatValue();
    }
    if(theOscMessage.checkAddrPattern("/1/fader1")==true) {
        yAmplify = theOscMessage.get(0).floatValue(); // get the second osc argument
        println(" values: "+yAmplify);
    }
}
```

Arduino sketch (arduino_client).

A sketch is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board. When the hardware part is done open up the Arduino application and type the below code into the editor. Compile the sketch and install it on the Arduino.

```
#include <SoftwareServo.h>
// include the library code:
#include <LiquidCrystal.h>
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(7, 8, 4, 5, 6, 13);

SoftwareServo servol;
SoftwareServo servo2;
SoftwareServo servo3;
// initializes servo
int servolPos = 90;
int servo2Pos = 90;
int servo3Pos = 90;
int data = 0;
char str[1];

void setup() {
    // set up the LCD's number of rows and columns:
    lcd.begin(16, 2);
    lcd.print("Delta robot");

    pinMode(13, OUTPUT);

    servol.attach(10); // attaches the servo on pin 10 to the servo object
    servo2.attach(11); // attaches the servo on pin 11 to the servo object
    servo3.attach(12); // attaches the servo on pin 12 to the servo object
    //pinmode
    Serial.begin(115200);
}

void loop() {
    if (Serial.available() > 0) {
```

```
// read the incoming byte:
data = Serial.read();

if (data == 255) {
  delay(7);
  if (Serial.available() > 0) {
    serv1Pos = Serial.read();
    if (serv1Pos < 30)
      serv1Pos = 30;
    serv1.write(serv1Pos);
  }
}

  if (data == 254) {
    delay(7);
    if (Serial.available() > 0) {
      servo2Pos = Serial.read();
      if (servo2Pos < 30)
        servo2Pos = 30;
      servo2.write(servo2Pos);
    }
  }

  if (data == 253) {
    delay(7);
    if (Serial.available() > 0) {
      servo3Pos = Serial.read();
      if (servo3Pos < 30)
        servo3Pos = 30;
      servo3.write(servo3Pos);
    }
  }
}

lcd.setCursor(0, 1);
itoa(servo3Pos, str, 10);
lcd.print(str);
lcd.setCursor(5, 1);
itoa(servo2Pos, str, 10);
lcd.print(str);
lcd.setCursor(10, 1);
itoa(serv1Pos, str, 10);
lcd.print(str);

SoftwareServo::refresh();
}
```

I also have published a video about my delta robot on youtube. You can see it at:

<http://www.youtube.com/watch?v=wCG3QVIlqsl>

Table of figures

Figure 1 - technical drawing from U.S. Patent n. 4,976,582.....	4
Figure 2 - Kinematic scheme of the delta robot.....	6
Figure 3 - some key parameters of our robot's geometry	7
Figure 4 – how to find point J1	8
Figure 5 – how to find theta1	8
Figure 6 – how to find theta2 and theta3	9
Figure 7 - forward kinematics, key parameters.....	10
Figure 8 - intersection of three spheres	11
Figure 9 - how to find J1', J2', J3' coordinates	11
Figure 10 - use cases.....	13
Figure 11 - system architecture.....	14
Figure 12 - class diagram	14
Figure 13 - sequence diagrams.....	15
Figure 14 - TouchOSC Editor.....	22
Figure 15 - TouchOSC options	23
Figure 16 – servo motors interconnection	24
Figure 17 - LCD display interconnection.....	24
Figure 18 - my delta robot.....	25

Bibliography

1. **Wikipedia**. Delta robot. *Wikipedia*. [Online] [Riportato: 7 September 2010.]
http://en.wikipedia.org/wiki/Delta_robot.
2. **Mzavatsky**. Delta robot kinematics. *Trossenrobotics*. [Online] [Riportato: 7 September 2010.]
<http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/>.
3. **Mjcbuin**. Delta bot. *Letsmakerobots*. [Online] [Riportato: 7 September 2010.]
<http://letsmakerobots.com/node/10577>.
4. **Touchosc**. Touchosc. *hexler.net*. [Online] [Riportato: 12 September 2010.]
<http://hexler.net/software/touchosc>.
5. **Kionix**. Kionix. *kionix.com*. [Online] 23 April 2007. [Riportato: 18 September 2010.]
<http://www.kionix.com/App-Notes/AN013%20Double%20integration.pdf>.