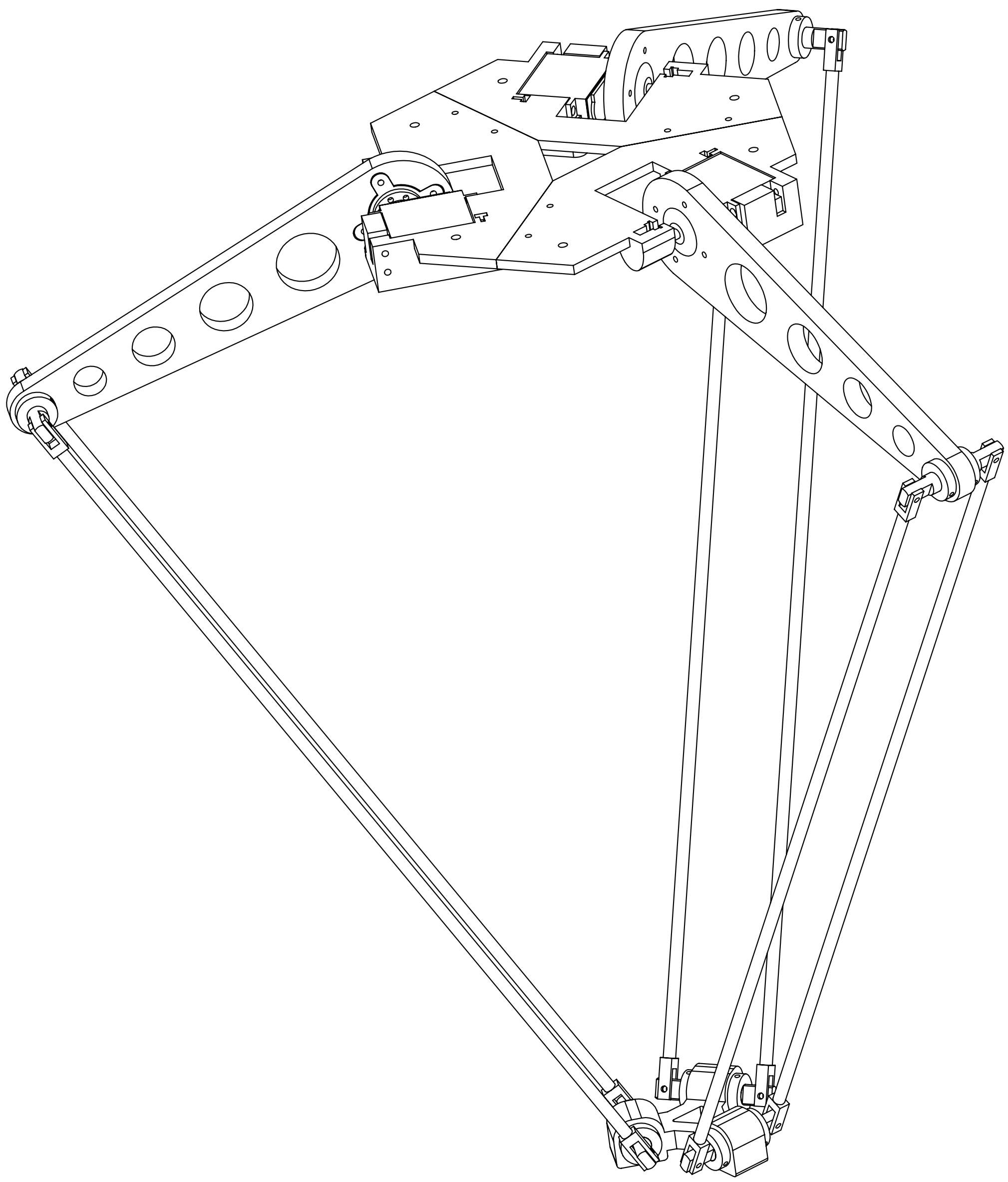


UPAT Delta Robot version 1.0

Assembly and Operations Manual

University of Patras
Electrical & Computer Engineering Department
Laboratory of Automation & Robotics

This document and any other relevant information regarding this manipulator can be referenced as:
A. Tzes, G. Ntekoumes, K. Gkountas, and K. Giannousakis, 'Design and control of an open-source long-reach Delta-robot for attachment to an Unmanned Aerial Vehicle', Technical Report, December 2016 (URL: https://github.com/UPatras-ANeMoS/UPAT-Delta-Robot)

Chapter 1

UPAT Delta Robot General Information

1.1 UPAT Development Team

The research team at University of Patras (UPAT) comprised of Professor Anthony Tzes, senior undergraduate student Georgios Ntekoumes, and graduate students Konstantinos Gkountas and Konstantinos Giannousakis, designed, implemented, and controlled a 3 DoF robot arm relying on the delta-arm configuration. This robotic arm, to be called hereafter UPAT delta, to be used as part of the deliverable WP3 and for testing purposes in WP5 of EU's Horizon 2020 Aeroworks program in which UPAT participates.

1.2 Robot Overview

UPAT's Delta robot Version 1.0, is a long-reach manipulator, weighs 1 Kgr, has a workspace of 20cm x 20cm x 10cm (x, y, z), has three Hitec HS-645MG actuators, requires at a maximum 18 W for operation (at a 12 Volt rated voltage), and can carry a payload of 300 gr.

It consists of a fixed base, 3 upper legs, 3 lower legs and the moving platform which is parallel to the fixed base. Analytically, the parts are, as shown in Figure 1.1.

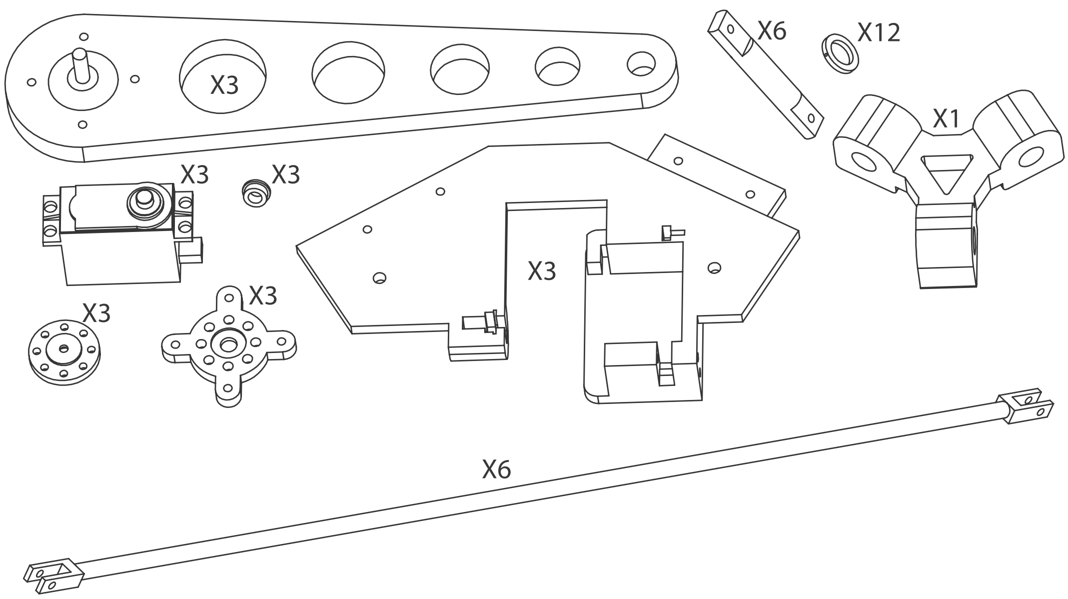


Figure 1.1: UPAT Delta Robot Componets

1.3 Robot Mechanical Components

- Three 3D-printed parts composing the fixed upper base, shown in Figure 1.2.

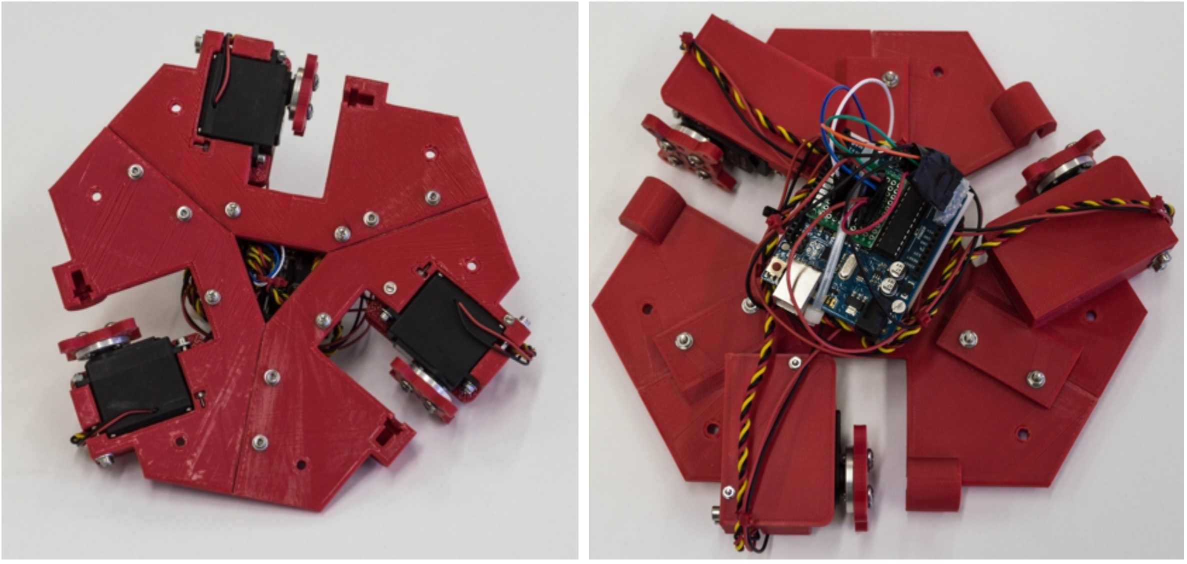


Figure 1.2: UPAT Delta Robot Upper Base

- Three servos, servo hubs and servo horns, shown in Figure 1.3 (left).
- Three 3D-pinted upper legs with their three attached bearings, shown in Figure 1.3 (right).

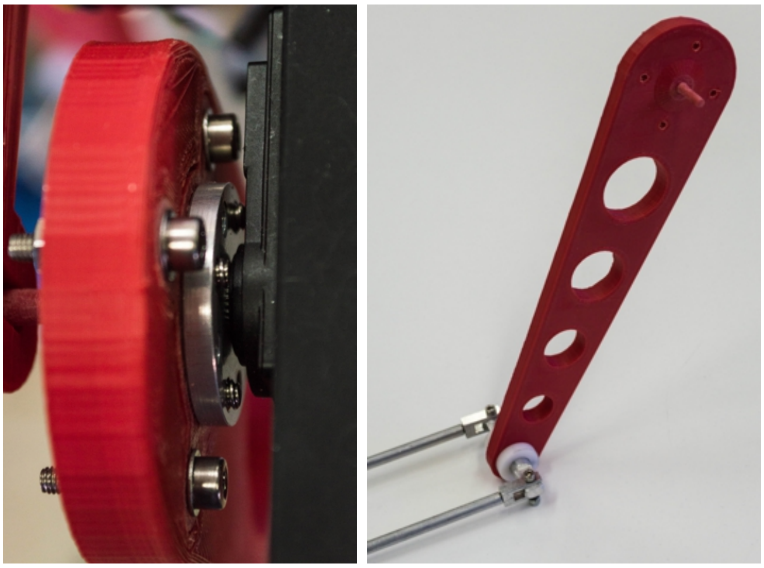


Figure 1.3: UPAT Delta Robot servo & 3D Upper Leg

- Three aluminum lower legs (corresponding to the 4-bar parallelogram mechanism) and 12 custom-made rings, shown in Figure 1.4 (left)
- 3D-printed moving platform, shown in Figure 1.4 (right).

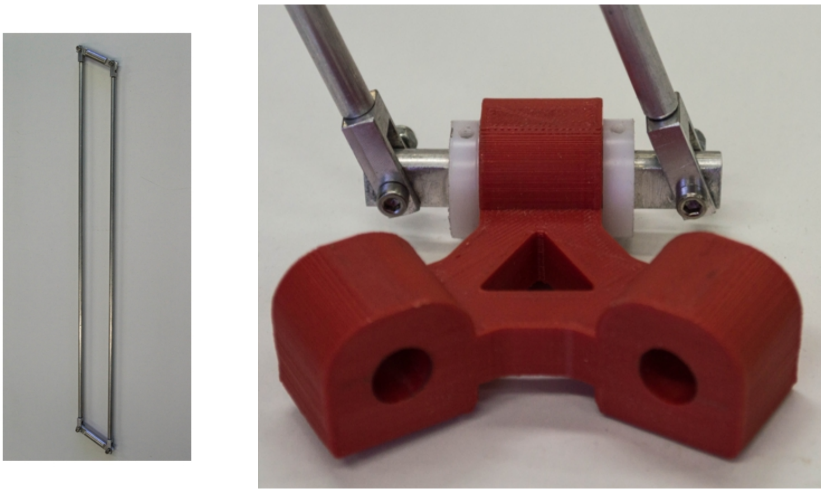


Figure 1.4: UPAT Delta Robot Parallelogram & 3D Lower Base

1.4 Electrical and Electronic Components

For the application of control commands, the Phidget Advanced Servo board is used; this board can connect up to 8 servos and demands a power voltage in the range 6-15 Volt. Each servo utilizes an internal P-control law employing Mitsubishi's M51660L IC. Rather than relying on the internal P-controller, an outer loop controller can also be employed. For this reason, there is direct access to the potentiometer attached to the servo. The resistance is measured via a simple circuit using a single supply LM324-N operational amplifier. The voltage corresponding to the measured voltage is measured by the 10-bit Analog-to-Digital Converter embedded on an Arduino UNO Rev 3 microcontroller.

Both the Arduino and Phidget boards are connected to a computer via their USB-ports, as shown in Figure 1.5, while the servos are powered via the Phidget-board.

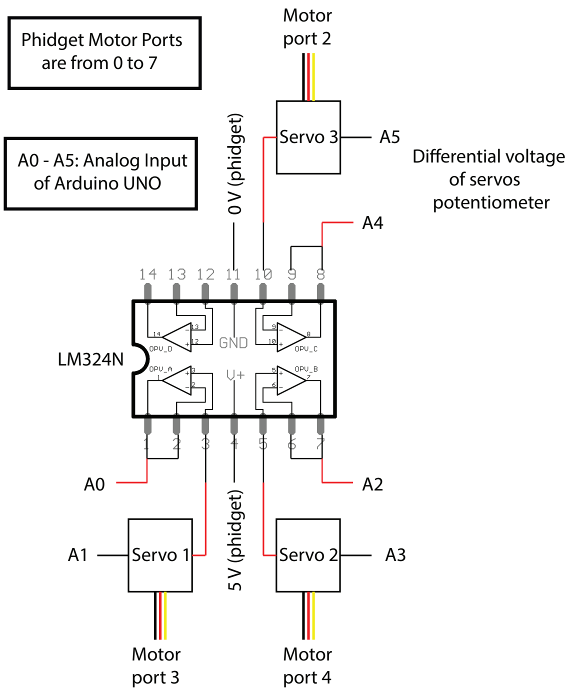


Figure 1.5: UPAT Delta Robot Electronics Schematic

1.5 Robot Attachment & Dimensions

For the robot's attachment, six M4 symmetrically placed screws can be placed in their corresponding holes, as shown in Figure 1.6. In the sequel, the robot's coordinate-system (X_B, Y_B, Z_B) attached at its base is shown in the same Figure, with its origin $(\{x, y, z\} = \{0, 0, 0\})$ at the upper base's symmetrical point.

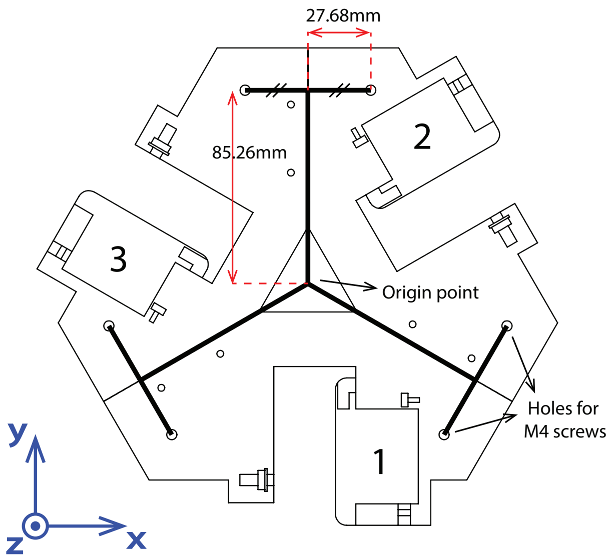


Figure 1.6: UPAT delta robot attachment & dimensions

In Figure 1.7 the robot links' dimensions are shown

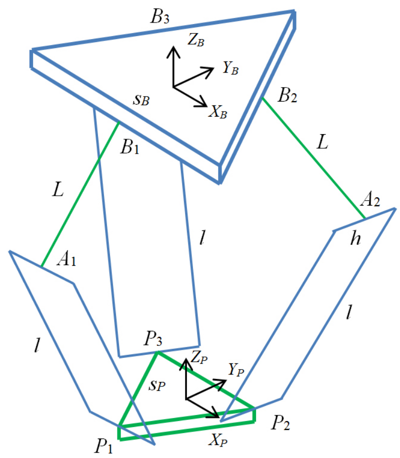


Figure 1.7: UPAT Delta Robot Link Dimensions

	Description	**mm**
s_B	Upper base equilateral triangle side	300
s_P	Lower platform equilateral triangle side	50
L	Upper leg's length	200
l	Lower leg's parallelogram length	510
h	Lower leg's parallelogram width	50

Chapter 2

UPAT Delta Robot Assembly

Besides for the Hitec HS-645MG servos and the flanged ball bearings MF84ZZ, and the screws, the remaining 3D-parts are are made of ABS-plastic, or of aluminum. In the sequel, the assembly of the robot is presented in a step-by-step basis.

1. The next figure shows how to assemble the upper (fixed) base of the robot, composed of three identical parts snapped and tightened with 6 screws.

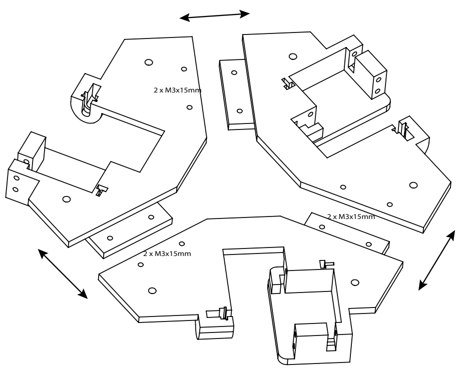


Figure 2.1: UPAT Delta Robot Assembly #1

2. In each part of the base(A), as shown in the next figure, we place the servo(B), the servo hub(C) and the servo horn(D) on it, in the order shown.

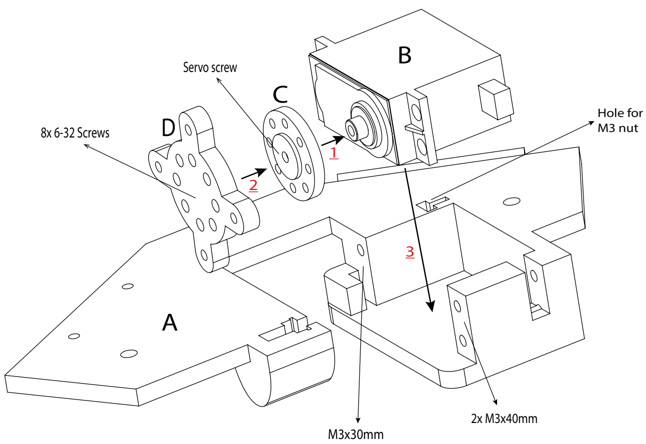


Figure 2.2: UPAT Delta Robot Assembly #2

3. Subsequently, one bearing is attached at the axis of each upper leg, as shown in the following step (1), followed by placement of the upper leg in the designated space of the fixed base using the servo horn (2). For the attachment of the bearing, it is important to be in position for placement in its groove.

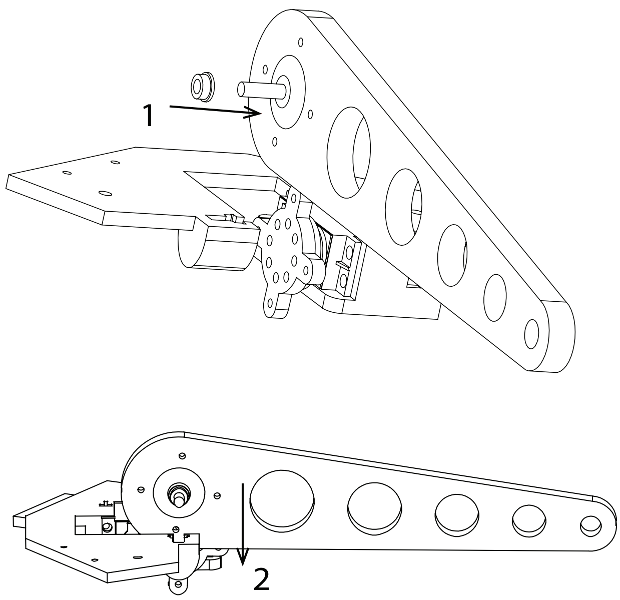


Figure 2.3: UPAT Delta Robot Assembly #3

4. The upper leg with the servo horn is attached at the base (3), followed by the proper screwing (4) of the upper leg to the servo horn and the remaining parts of the base. Since the bearing will remain fixed in its position, it is imperative to the test the limits from $-90°$ to $90°$ of the servo.

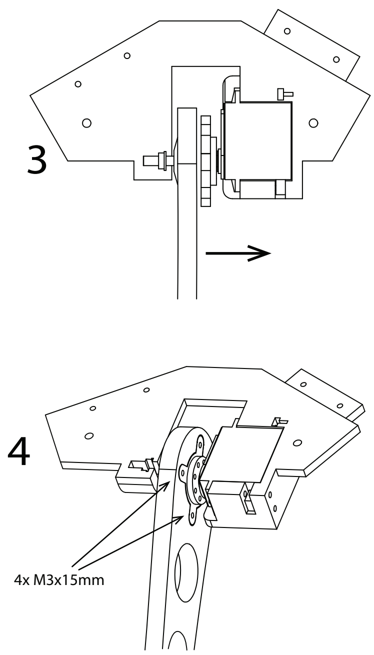


Figure 2.4: UPAT Delta Robot Assembly #4

5. Placement of the rotational joint of the upper leg, along with the placement of the rings used for alignment.

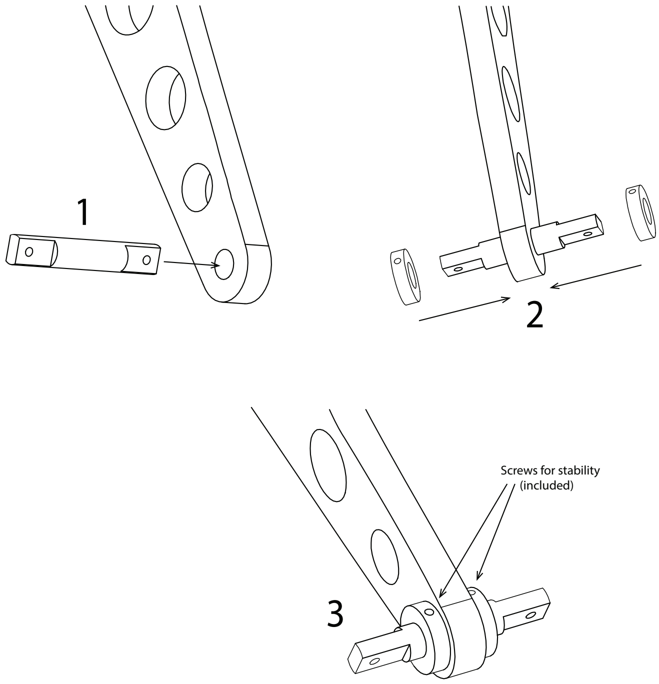


Figure 2.5: UPAT Delta Robot Assembly #5

6. Construction of each lower leg of the robot's parallelogram; there are 6 legs each tapped at both sides.

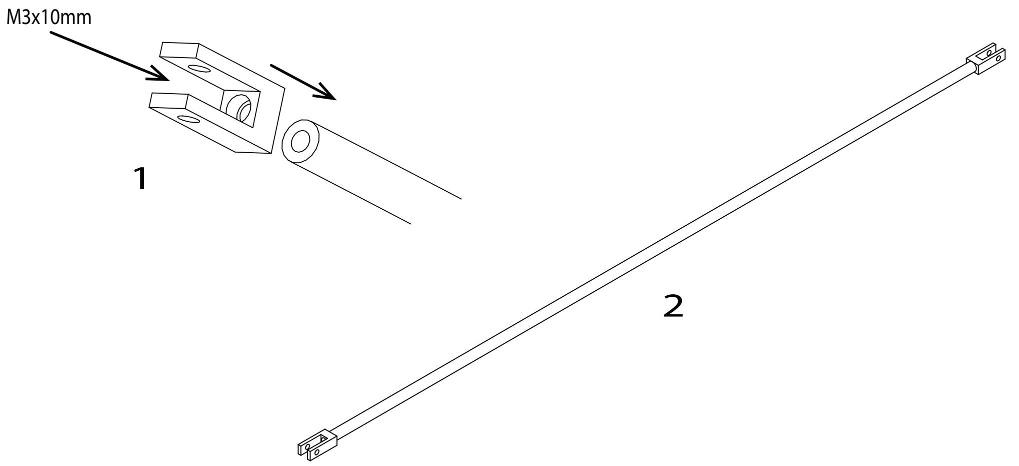


Figure 2.6: UPAT Delta Robot Assembly #6

7. Assembly of the robot's moving platform, using three rotational joints with their assorted rings

8. Attachment of the moving platform to the legs of the parallelogram and the joint of the upper legs.

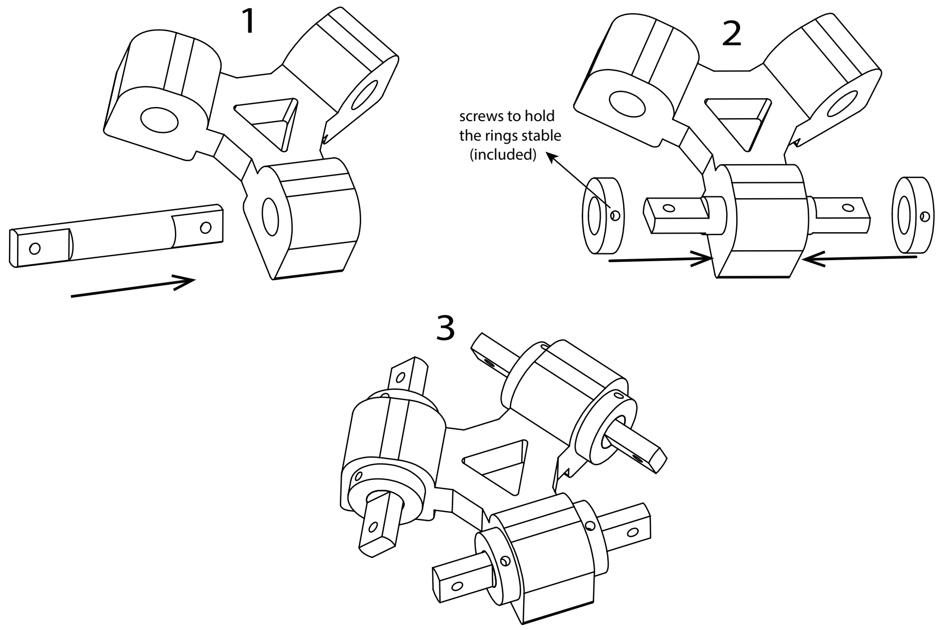


Figure 2.7: UPAT Delta Robot Assembly #7

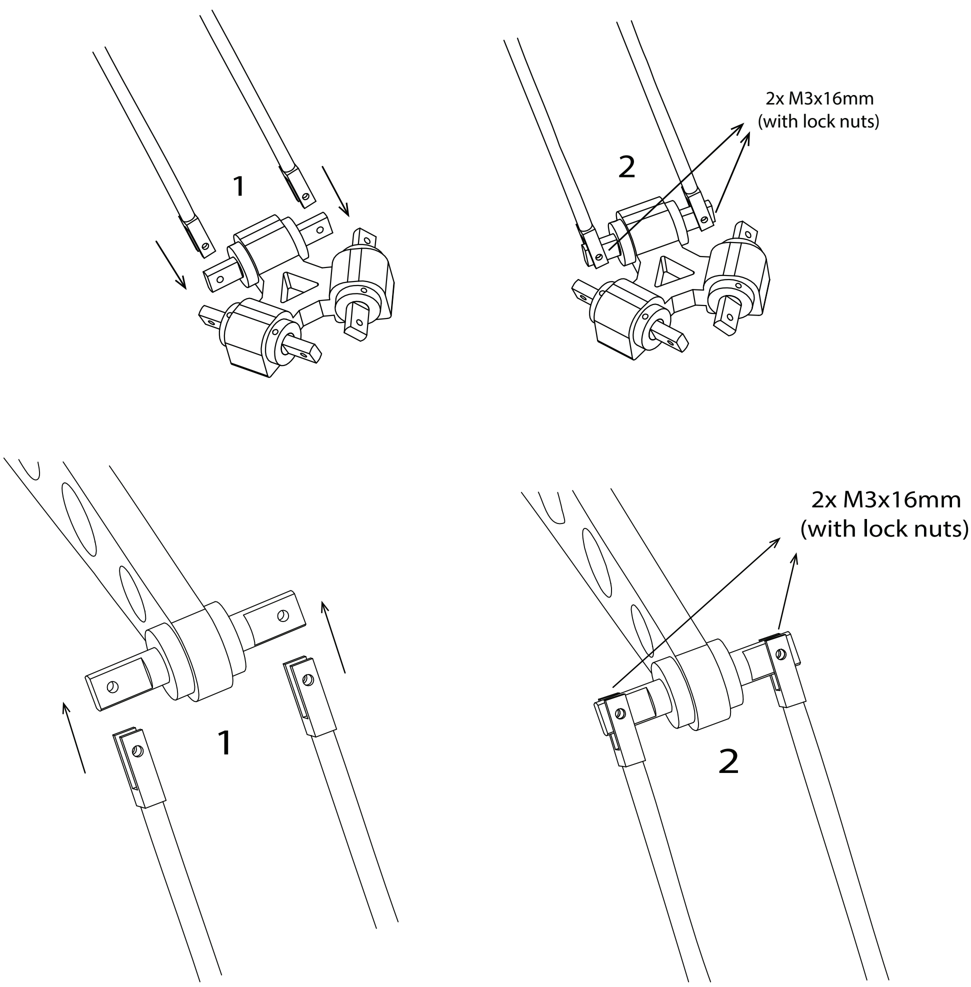


Figure 2.8: UPAT Delta Robot Assembly #8

Chapter 3

Controller Software in ROS

3.1 Package Installation

The outer loop controller was written in C (codes in Github) and tested in a computer running Ubuntu 16.04 LTS. The Robot Operating System (ROS) must be installed along with the libraries to operate with the Phidget Advanced Servo.

The first step is the creation of workspace delta in the directory ∼/home/. The ROS-driver publisher (arduino) node and the subscriber (phidget) node have to be placed in the aforementioned workspace. Following this step, the program can be executed.

A terminal is opened and the packages in the delta workspace are built:

```
$ cd ~ /delta
$ catkin_make
```

The program can be executed in two alternative ways:

1. There is a launch file, that runs automatically the nodes and opens a new terminal where the user can interact and control the movement of the robot. For this option, the user excutes the following command in a terminal:

   ```
   $ roslaunch phidget delta.launch
   ```

2. The alternative option is to execute the nodes manually.

   - In a terminal type:

     ```
     $ roscore
     ```

   - Open a new terminal and run the command:

     ```
     $ rosrun arduino arduino
     ```

     This is the publisher who provides the feedback of the servos and must be executed prior to the launch of the other node.

   - Open a new terminal and type the command:

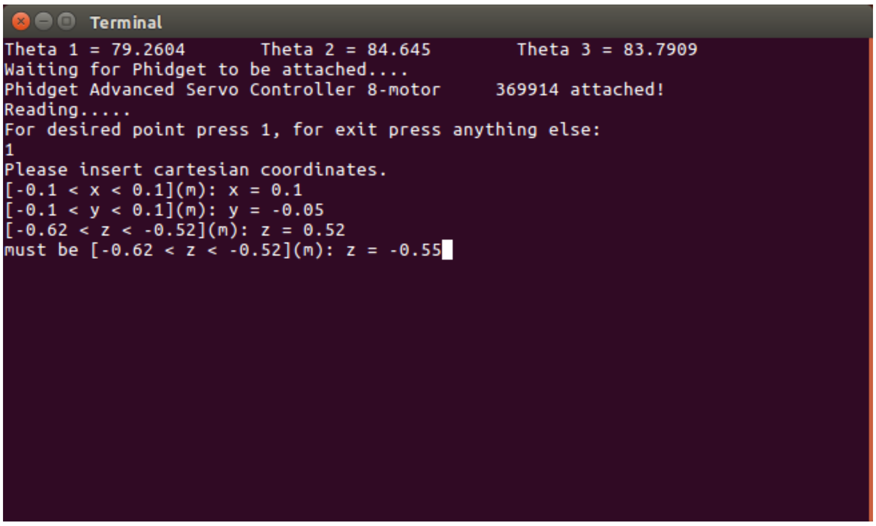
     ```
     $ rosrun phidget phidget
     ```

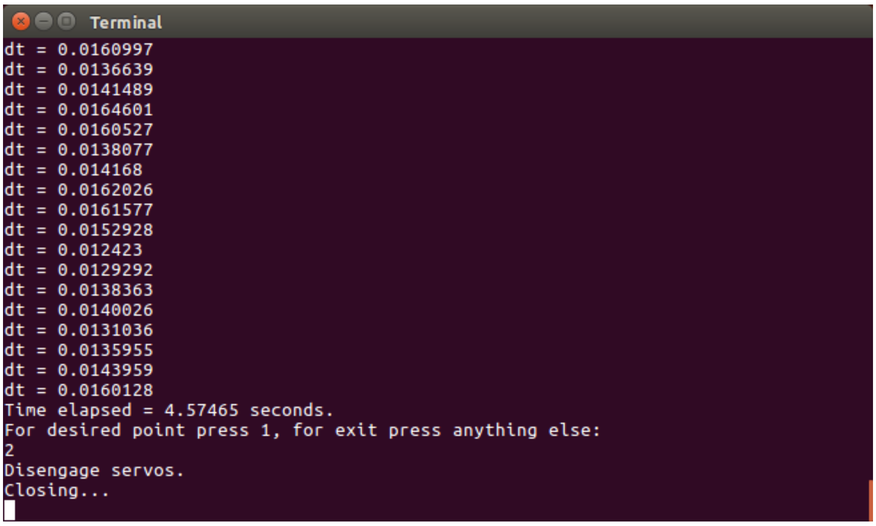
     This command runs the main node and this is the terminal from which the user interacts with the robot.

When the both nodes operate, the robot's software kernel engages its servos and awaits to get its first position command. In the last terminal there is interaction between the robot and the user. The user has the option of either providing the Cartesian coordinates where the robot's bottom (moving) platform is to move, or terminating the program.

If the user selects the first option, three (3) double numbers (one at a time) must be inserted, corresponding to the platforms x, y, z coordinates expressed in meters. This continues, until the user terminates the program, in which case the robots moves to its initial position and disengages its servos.

The next Figures highlight the interaction terminal.





3.2 Simulation Studies

From kinematic analysis, as shown in Figure 1.7 we have:

$$ {}^B\mathbf{B}_1 = \begin{Bmatrix} 0 \\ -w_B \\ 0 \end{Bmatrix} \qquad {}^B\mathbf{B}_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2}w_B \\ \frac{1}{2}w_B \\ 0 \end{Bmatrix} \qquad {}^B\mathbf{B}_3 = \begin{Bmatrix} -\frac{\sqrt{3}}{2}w_B \\ \frac{1}{2}w_B \\ 0 \end{Bmatrix} $$

$$ {}^P\mathbf{P}_1 = \begin{Bmatrix} 0 \\ -u_P \\ 0 \end{Bmatrix} \qquad {}^P\mathbf{P}_2 = \begin{Bmatrix} \frac{s_P}{2} \\ w_P \\ 0 \end{Bmatrix} \qquad {}^P\mathbf{P}_3 = \begin{Bmatrix} -\frac{s_P}{2} \\ w_P \\ 0 \end{Bmatrix} $$

$$ \{{}^B\mathbf{B}_i\} + \{{}^B\mathbf{L}_i\} + \{{}^B\mathbf{I}_i\} = \{{}^B\mathbf{P}_P\} + \left[{}^B_P\mathbf{R}\right]\{{}^P\mathbf{P}_i\} = \{{}^B\mathbf{P}_P\} + \{{}^P\mathbf{P}_i\} \qquad i = 1, 2, 3 $$

where $\left[{}^B_P\mathbf{R}\right] = [\mathbf{I}_3]$, since no rotations are allowed by the Delta Robot.

$$ {}^B\mathbf{L}_1 = \begin{Bmatrix} 0 \\ -L\cos\theta_1 \\ -L\sin\theta_1 \end{Bmatrix} \qquad {}^B\mathbf{L}_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2}L\cos\theta_2 \\ \frac{1}{2}L\cos\theta_2 \\ -L\sin\theta_2 \end{Bmatrix} \qquad {}^B\mathbf{L}_3 = \begin{Bmatrix} -\frac{\sqrt{3}}{2}L\cos\theta_3 \\ \frac{1}{2}L\cos\theta_3 \\ -L\sin\theta_3 \end{Bmatrix} $$

Substituting all above values into the vector-loop closure equations yields:

$$ {}^B\mathbf{I}_1 = \begin{Bmatrix} x \\ y + L\cos\theta_1 + a \\ z + L\sin\theta_1 \end{Bmatrix} \qquad {}^B\mathbf{I}_2 = \begin{Bmatrix} x - \frac{\sqrt{3}}{2}L\cos\theta_2 + b \\ y - \frac{1}{2}L\cos\theta_2 + c \\ z + L\sin\theta_2 \end{Bmatrix} \qquad {}^B\mathbf{I}_3 = \begin{Bmatrix} x + \frac{\sqrt{3}}{2}L\cos\theta_3 - b \\ y - \frac{1}{2}L\cos\theta_3 + c \\ z + L\sin\theta_3 \end{Bmatrix} $$

where:
$$ \begin{aligned} a &= w_B - u_P \\ b &= \frac{s_P}{2} - \frac{\sqrt{3}}{2}w_B \\ c &= w_P - \frac{1}{2}w_B \end{aligned} $$

And the three constraint equations yield the kinematics equations for the Delta Robot:

$$ \begin{aligned} 2L(y+a)\cos\theta_1 + 2zL\sin\theta_1 + x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 &= 0 \\ -L(\sqrt{3}(x+b) + y + c)\cos\theta_2 + 2zL\sin\theta_2 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2xb + 2yc - l^2 &= 0 \\ L(\sqrt{3}(x-b) - y - c)\cos\theta_3 + 2zL\sin\theta_3 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 - 2xb + 2yc - l^2 &= 0 \end{aligned} $$

The three absolute vector knee points are found using ${}^B\mathbf{A}_i = {}^B\mathbf{B}_i + {}^B\mathbf{L}_i, i = 1, 2, 3$:

$$ {}^B\mathbf{A}_1 = \begin{Bmatrix} 0 \\ -w_B - L\cos\theta_1 \\ -L\sin\theta_1 \end{Bmatrix} \qquad {}^B\mathbf{A}_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2}(w_B + L\cos\theta_2) \\ \frac{1}{2}(w_B + L\cos\theta_2) \\ -L\sin\theta_2 \end{Bmatrix} \qquad {}^B\mathbf{A}_3 = \begin{Bmatrix} -\frac{\sqrt{3}}{2}(w_B + L\cos\theta_3) \\ \frac{1}{2}(w_B + L\cos\theta_3) \\ -L\sin\theta_3 \end{Bmatrix} $$

3.2.1 Forward Kinematics

The 3-dof Delta Robot forward position kinematics (FPK) problem is stated: Given the three actuated joint angles $\boldsymbol{\theta} = \{\theta_1 \quad \theta_2 \quad \theta_3\}^T$, calculate the resulting Cartesian position of the moving platform control point (the origin of P), ${}^B\mathbf{P}_P = \{x \quad y \quad z\}^T$.

Thanks to the translation-only motion of the 3-dof Delta Robot, there is a straightforward analytical solution for which the correct solution set is easily chosen. Since $\boldsymbol{\theta} = \{\theta_1 \quad \theta_2 \quad \theta_3\}^T$ are given, we calculate the three absolute vector knee points using ${}^B\mathbf{A}_i = {}^B\mathbf{B}_i + {}^B\mathbf{L}_i, i = 1, 2, 3$. Since we know that the moving platform orientation is constant, always horizontal with $\left[{}^B_P\mathbf{P}\right] = [\mathbf{I}_3]$, we define three virtual sphere centers ${}^B\mathbf{A}_{iv} = {}^B\mathbf{A}_i - {}^P\mathbf{P}_i, i = 1, 2, 3$:

$$
{}^B\mathbf{A}_{1v} = \left\{ \begin{matrix} 0 \\ -w_B - L\cos\theta_1 + u_P \\ -L\sin\theta_1 \end{matrix} \right\}
{}^B\mathbf{A}_{2v} = \left\{ \begin{matrix} \frac{\sqrt{3}}{2}(w_B + L\cos\theta_2) - \frac{s_P}{2} \\ \frac{1}{2}(w_B + L\cos\theta_2) - w_P \\ -L\sin\theta_2 \end{matrix} \right\}
{}^B\mathbf{A}_{3v} = \left\{ \begin{matrix} -\frac{\sqrt{3}}{2}(w_B + L\cos\theta_3) + \frac{s_P}{2} \\ \frac{1}{2}(w_B + L\cos\theta_3) - w_P \\ -L\sin\theta_3 \end{matrix} \right\}
$$

Then the Delta Robot FPK solution is the intersection point of three known spheres. Let a sphere be referred as a vector center point c and scalar radius r, (c, r). Therefore, the FPK unknown point $\{{}^B\mathbf{P}_P\}$ is the intersection of the three known spheres:

$$
(\{{}^B\mathbf{A}_{1v}\}, l) \qquad (\{{}^B\mathbf{A}_{2v}\}, l) \qquad (\{{}^B\mathbf{A}_{3v}\}, l)
$$

3.2.2 Inverse Kinematics

The 3-dof Delta Robot inverse position kinematics (IPK) problem is stated: Given the Cartesian position of the moving platform control point (the origin of P), ${}^B\mathbf{P}_P = \{x \quad y \quad z\}^T$, calculate the three required actuated revolute joint angles $\boldsymbol{\theta} = \{\theta_1 \quad \theta_2 \quad \theta_3\}^T$.

This solution may be done geometrically/trigonometrically. However, we will now accomplish this IPK solution analytically, using the three constraint equations applied to the vector loop-closure equations (derived previously). The three independent scalar IPK equations are of the form:

$$
E_i \cos\theta_i + F_i \sin\theta_i + G_i = 0 \qquad i = 1, 2, 3
$$

where:

$$
\begin{aligned}
E_1 &= 2L(y + a) \\
F_1 &= 2zL \\
G_1 &= x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2
\end{aligned}
$$

$$
\begin{aligned}
E_2 &= -L(\sqrt{3}(x + b) + y + c) & E_3 &= L(\sqrt{3}(x - b) - y - c) \\
F_2 &= 2zL & F_3 &= 2zL \\
G_2 &= x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2(xb + yc) - l^2 & G_3 &= x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2(-xb + yc) - l^2
\end{aligned}
$$

3.3 Experimental Studies

In the ensuing experimental study, the robot is commanded to move its “engaged” point to the point {0 , 0.05 , -0.6}, then to the {-0.06 , -0.05 , -0.57} and final to the {0.06 , -0.05 , -0.54}. The outer loop PID-controller executes throughout this scenario; the robot is assumed to be in the neighborhood of each point if the actual angles measured by the potentiometer are within one degree 1° difference from the desired ones. In the occurrence of such an event, the robot waits the command to go to an another point or to terminate its operation. The selected gains of the PID-controller were computed so as to assure a smooth transition from point-to-point maneuvres without any 'large' overhoots in the event of carrying a payload. The user is encouraged not to alter these gains, despite the rather slow response of the manipulator.

In the sequel, we provide the experimental and simulation results for the aforementioned trajectory focusing only on the accuracy of the achieved final points.

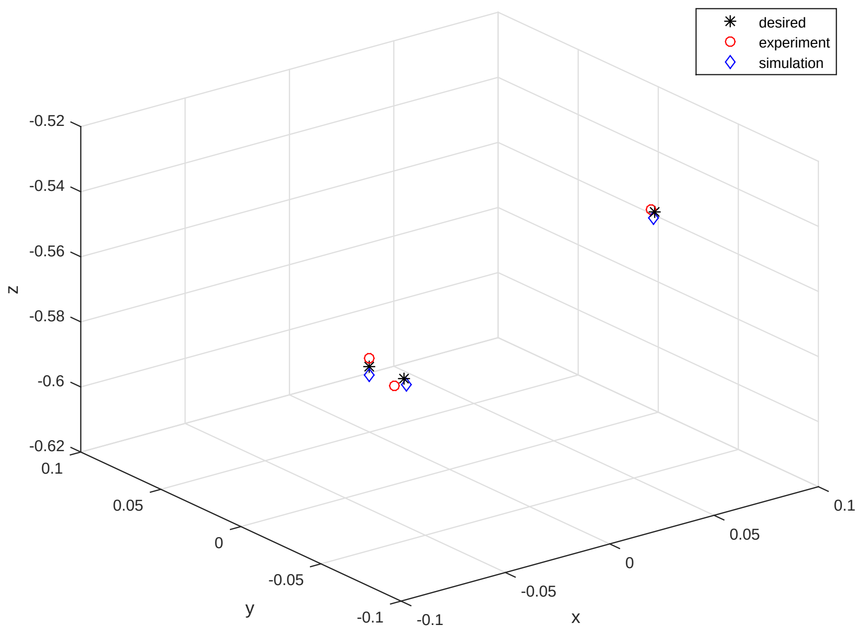
Experiment Results

X(desired)	**Y(desired)**	**Z(desired)**	**X(actual)**	**Y(actual)**	**Z(actual)**
0.00	0.05	-0.60	0.0013	0.0515	-0.5978
-0.06	-0.05	-0.57	-0.0639	-0.0488	-0.5720
0.06	-0.05	-0.54	0.0621	-0.0447	-0.5408

Simulation Results

X(desired)	**Y(desired)**	**Z(desired)**	**X(simulation)**	**Y(simulation)**	**Z(simulation)**
0.00	0.05	-0.60	-0.0004	0.0497	-0.6022
-0.06	-0.05	-0.57	-0.0593	-0.0499	-0.5722
0.06	-0.05	-0.54	0.0598	-0.0492	-0.5421

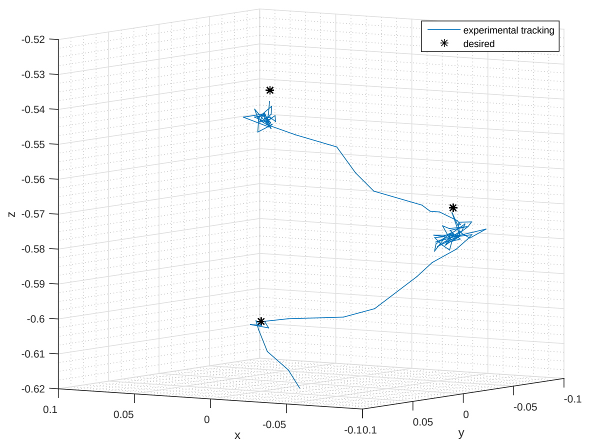
In Figure 3.1, we present for visualization purposes, the achieved results for the noted trajectory for the UPatras manipulator



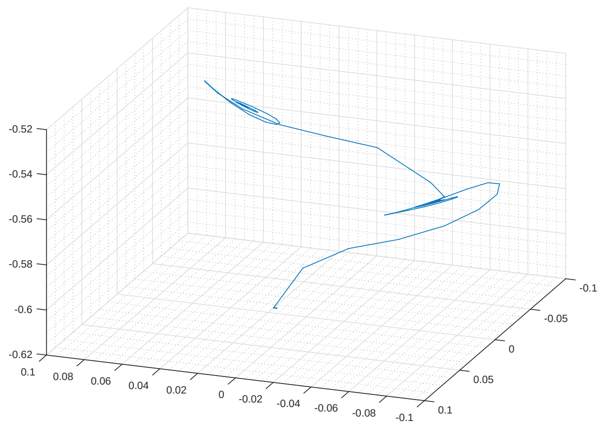
(a) 3D view

Figure 3.1: UPatras Delta Robot Experimental & Simulation Study

The trajectories of the robot in the experiment and in the simulation are presented in Figures 3.2a and 3.2b, respectively.



(a) Experimental trajectory



(b) Simulation trajectory

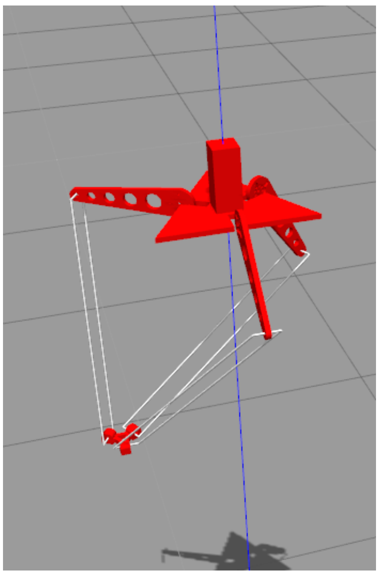
Figure 3.2

3.4 UPAT Delta Robot simulator in Gazebo

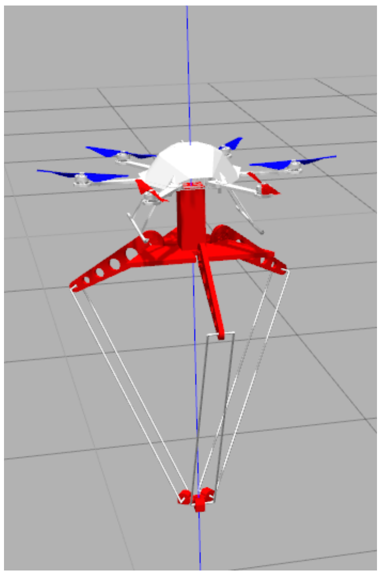
A Gazebo-based simulator for: a) a standalone UPAT Delta arm, and b) an attached arm at the Ascending Technologies Neo UAV has been posted in the group's Github, ;the code is open-source (see Appendix B).

The manipulator is assumed to be attached via a 5 cm hollow cylinder at the base of the UAV. This cylinder is primarily used for visualization purposes, and it is up to the designers to decide on the proper attachment method.

Typical screenshots for the: a) the standalone arm, and b) the attached arm to the UAV are presented in Figures 3.3a and 3.3b, respectively.



(a) Standalone UPAT Delta arm



(b) UPAT Delta arm attached to UAV

Figure 3.3

In Figure 3.4 the difference in the achieved arm's trajectory for a clamped (standalone) arm for the previous trajectory and the one for the same arm coupled to a hovering UAV (Ascending Technology Neo) using its existing controller is provided.

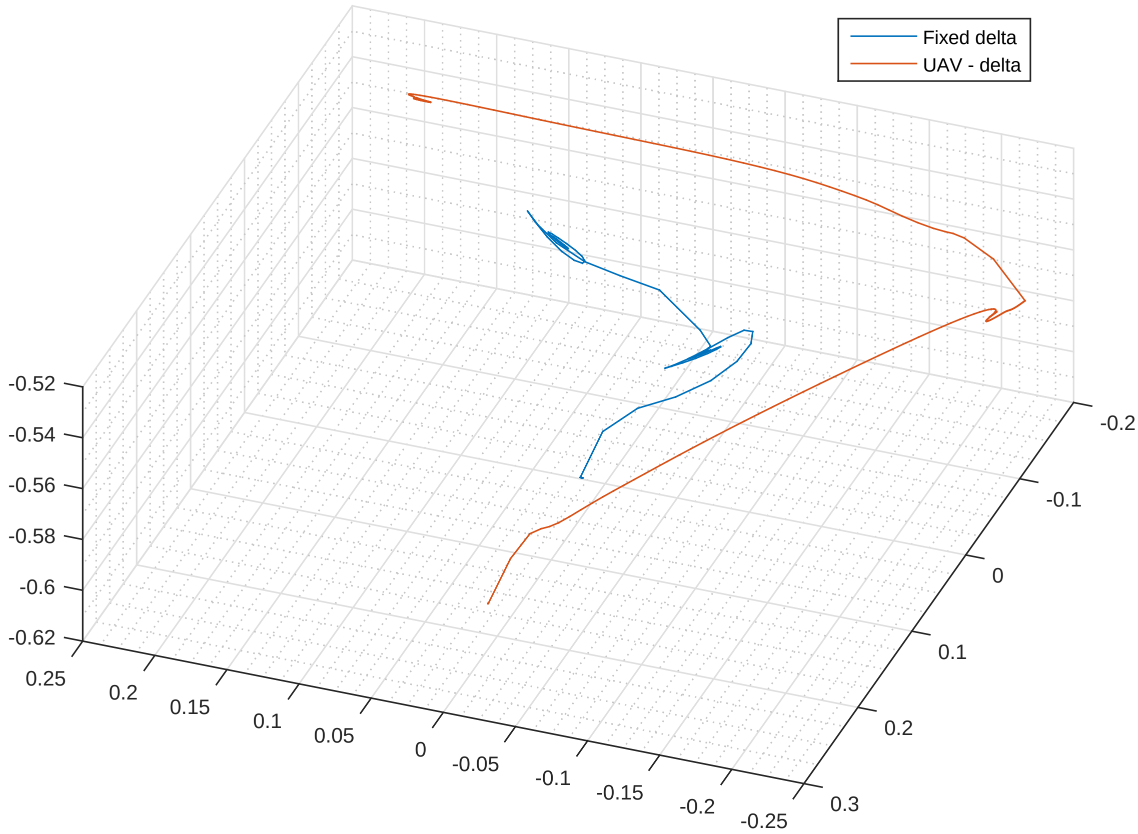


Figure 3.4: Gazebo Simulation

Appendix A - Kinematics codes

- Forward Kinematics

```
double sB = 0.3; double sP = 0.05; double L = 0.2; double l = 0.51;              //delta dimensions
void ForwardKinematics(double &x, double &y, double &z, double &theta1, double &theta2, double &theta3)
{
        double wB = (sqrt(3) / 6)*sB; double uB = (sqrt(3) / 3)*sB; double wP = (sqrt(3) / 6)*sP;
        double uP = (sqrt(3) / 3)*sP;
        // Points Av
        double x1 = 0; double y1 = -wB - L*cos(theta1*pi / 180) + uP; double z1 = -L*sin(theta1*pi / 180);
        double x2 = (sqrt(3) / 2)*(wB + L*cos(theta2*pi / 180)) - sP / 2;
        double y2 = (wB + L*cos(theta2*pi / 180)) / 2 - wP; double z2 = -L*sin(theta2*pi / 180);
        double x3 = -(sqrt(3) / 2)*(wB + L*cos(theta3*pi / 180)) + sP / 2;
        double y3 = (wB + L*cos(theta3*pi / 180)) / 2 - wP;     double z3 = -L*sin(theta3*pi / 180);

        if (z1 == z2 && z1 == z3){
                double a = 2 * (x3 - x1); double b = 2 * (y3 - y1);
                double c = -pow(x1, 2) - pow(y1, 2) + pow(x3, 2) + pow(y3, 2);
                double d = 2 * (x3 - x2); double e = 2 * (y3 - y2);
                double f = -pow(x2, 2) - pow(y2, 2) + pow(x3, 2) + pow(y3, 2);

                x = (c*e - b*f) / (a*e - b*d);
                y = (a*f - c*d) / (a*e - b*d);

                double A = 1; double B = -2 * z1;
                double C = pow(z1, 2) - pow(l, 2) + pow(x - x1, 2) + pow(y - y1, 2);
                double z_1 = (-B + sqrt(pow(B, 2) - 4 * A*C)) / (2 * A);
                double z_2 = (-B - sqrt(pow(B, 2) - 4 * A*C)) / (2 * A);

                if (z_1 < z_2){ z = z_1; }
                else { z = z_2; }

        }

        else if (z1 == z3 && z1 != z2){
                double ze = z3;
                double a11 = 2 * (x3 - x1); double a12 = 2 * (y3 - y1);
                double a21 = 2 * (x3 - x2); double a22 = 2 * (y3 - y2); double a23 = 2 * (ze - z2);
                double b1 = -pow(x1, 2) - pow(y1, 2) + pow(x3, 2) + pow(y3, 2);
                double b2 = -pow(x2, 2) - pow(y2, 2) - pow(z2, 2) + pow(x3, 2) + pow(y3, 2) + pow(ze, 2);

                double a1 = (b2 / a21) - (b1 / a11); double a2 = (a22 / a21) - (a12 / a11);
                double a3 = (a23 / a21); double a4 = -a3 / a2; double a5 = a1 / a2;
                double a6 = (-a12*a4) / a11; double a7 = (b1 - a12*a5) / a11;

                double a = pow(a4, 2) + pow(a6, 2) + 1;
                double b = 2 * a6*(a7 - x1) + 2 * a4*(a5 - y1) - 2 * ze;
                double c = a7*(a7 - 2 * x1) + a5*(a5 - 2 * y1) + pow(x1, 2) + pow(y1, 2) + pow(ze, 2) - pow(l, 2);

                double z_1 = (-b + sqrt(pow(b, 2) - 4 * a*c)) / (2 * a);
                double z_2 = (-b - sqrt(pow(b, 2) - 4 * a*c)) / (2 * a);

                if (z_1 < z_2){ z = z_1; }
                else { z = z_2; }

                x = a6*z + a7;
                y = a4*z + a5;

        }

        else if (z2 == z3 && z1 != z2){
                double ze = z3;
                double a11 = 2 * (x3 - x1); double a12 = 2 * (y3 - y1); double a13 = 2 * (ze - z1);
                double a21 = 2 * (x3 - x2); double a22 = 2 * (y3 - y2);
                double b1 = -pow(x1, 2) - pow(y1, 2) - pow(z1, 2) + pow(x3, 2) + pow(y3, 2) + pow(ze, 2);
                double b2 = -pow(x2, 2) - pow(y2, 2) + pow(x3, 2) + pow(y3, 2);

                double a1 = (a22 / a21) - (a12 / a11); double a2 = (a13 / a11);
                double a3 = (b2 / a21) - (b1 / a11); double a4 = a2 / a1; double a5 = a3 / a1;
                double a6 = (-a22*a4) / a21; double a7 = (b2 - a22*a5) / a21;

                double a = pow(a4, 2) + pow(a6, 2) + 1;
                double b = 2 * a6*(a7 - x1) + 2 * a4*(a5 - y1) - 2 * z1;
                double c = a7*(a7 - 2 * x1) + a5*(a5 - 2 * y1) + pow(x1, 2) + pow(y1, 2) + pow(z1, 2) - pow(l, 2);

                double z_1 = (-b + sqrt(pow(b, 2) - 4 * a*c)) / (2 * a);
                double z_2 = (-b - sqrt(pow(b, 2) - 4 * a*c)) / (2 * a);

                if (z_1 < z_2){ z = z_1; }
                else { z = z_2; }
```

```
            x = a6*z + a7;
            y = a4*z + a5;

        }

        else {
            double a11 = 2 * (x3 - x1); double a12 = 2 * (y3 - y1); double a13 = 2 * (z3 - z1);
            double a21 = 2 * (x3 - x2); double a22 = 2 * (y3 - y2); double a23 = 2 * (z3 - z2);
            double b1 = -pow(x1, 2) - pow(y1, 2) - pow(z1, 2) + pow(x3, 2) + pow(y3, 2) + pow(z3, 2);
            double b2 = -pow(x2, 2) - pow(y2, 2) - pow(z2, 2) + pow(x3, 2) + pow(y3, 2) + pow(z3, 2);

            double a1 = (a11 / a13) - (a21 / a23); double a2 = (a12 / a13) - (a22 / a23);
            double a3 = (b2 / a23) - (b1 / a13); double a4 = -a2 / a1; double a5 = -a3 / a1;
            double a6 = (-a21*a4 - a22) / a23; double a7 = (b2 - a21*a5) / a23;

            double a = pow(a4, 2) + pow(a6, 2) + 1;
            double b = 2 * a4*(a5 - x1) - 2 * y1 + 2 * a6*(a7 - z1);
            double c = a5*(a5 - 2 * x1) + a7*(a7 - 2 * z1) + pow(x1, 2) + pow(y1, 2) + pow(z1, 2) - pow(l, 2);

            double y_1 = (-b + sqrt(pow(b, 2) - 4 * a*c)) / (2 * a);
            double y_2 = (-b - sqrt(pow(b, 2) - 4 * a*c)) / (2 * a);

            double x_1 = a4*y_1 + a5;
            double x_2 = a4*y_2 + a5;
            double z_1 = a6*y_1 + a7;
            double z_2 = a6*y_2 + a7;

            if (z_1 < z_2){ x = x_1; y = y_1; z = z_1; }
            else { x = x_2; y = y_2; z = z_2; }

        }

}
```

- Inverse Kinematics

```
double sB = 0.3; double sP = 0.05; double L = 0.2; double l = 0.51;            //delta dimensions
void InverseKinematics(double &theta1, double &theta2, double &theta3, double &x, double &y, double &z)
{
        double wB = (sqrt(3) / 6)*sB; double uB = (sqrt(3) / 3)*sB; double wP = (sqrt(3) / 6)*sP;
        double uP = (sqrt(3) / 3)*sP;
        double a = wB - uP; double b = (sP / 2) - (sqrt(3)*wB) / 2; double c = wP - wB / 2;

        double E1 = 2 * L*(y + a);
        double F1 = 2 * z*L;
        double G1 = pow(x,2) + pow(y,2) + pow(z,2) + pow(a,2) + pow(L,2) + 2*y*a - pow(l,2);
        double E2 = -L*(sqrt(3)*(x + b) + y + c);
        double F2 = 2 * z*L;
        double G2 = pow(x, 2) + pow(y, 2) + pow(z, 2) + pow(b, 2) + pow(c, 2) + pow(L, 2) + 2 *(x*b + y*c) - pow(l, 2);
        double E3 = L*(sqrt(3)*(x - b) - y - c);
        double F3 = 2 * z*L;
        double G3 = pow(x, 2) + pow(y, 2) + pow(z, 2) + pow(b, 2) + pow(c, 2) + pow(L, 2) + 2 * (-x*b + y*c) - pow(l, 2);

        double t1_1 = (-F1 + sqrt(pow(E1, 2) + pow(F1, 2) - pow(G1, 2))) / (G1 - E1);
        double t1_2 = (-F1 - sqrt(pow(E1, 2) + pow(F1, 2) - pow(G1, 2))) / (G1 - E1);
        double t2_1 = (-F2 + sqrt(pow(E2, 2) + pow(F2, 2) - pow(G2, 2))) / (G2 - E2);
        double t2_2 = (-F2 - sqrt(pow(E2, 2) + pow(F2, 2) - pow(G2, 2))) / (G2 - E2);
        double t3_1 = (-F3 + sqrt(pow(E3, 2) + pow(F3, 2) - pow(G3, 2))) / (G3 - E3);
        double t3_2 = (-F3 - sqrt(pow(E3, 2) + pow(F3, 2) - pow(G3, 2))) / (G3 - E3);

        double th1_1 = 2 * atan(t1_1); double th1_2 = 2 * atan(t1_2);
        double th2_1 = 2 * atan(t2_1); double th2_2 = 2 * atan(t2_2);
        double th3_1 = 2 * atan(t3_1); double th3_2 = 2 * atan(t3_2);

        if (th1_1 <= (pi / 2) && th1_1 >= -(pi / 2)){ theta1 = (180 * th1_1) / pi; }
        else { theta1 = (180 * th1_2) / pi; }

        if (th2_1 <= (pi / 2) && th2_1 >= -(pi / 2)){ theta2 = (180 * th2_1) / pi; }
        else { theta2 = (180 * th2_2) / pi; }

        if (th3_1 <= (pi / 2) && th3_1 >= -(pi / 2)){ theta3 = (180 * th3_1) / pi; }
        else { theta3 = (180 * th3_2) / pi; }

}
```