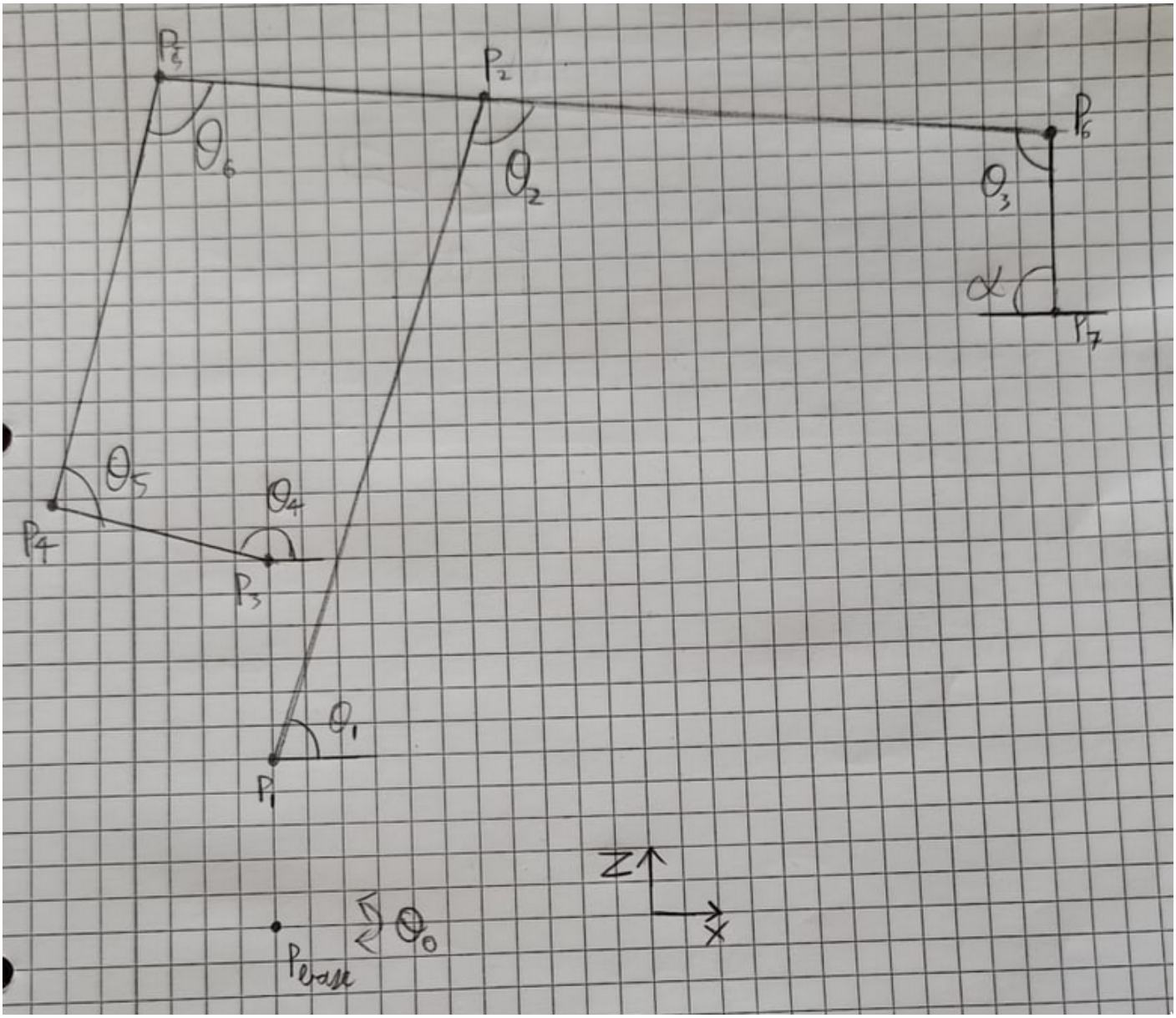
Kinematics

Defining terms

To do kinematics on the robotic arm, it first needs to be modelled in a way to do calculations on the different joint angles and positions. This is done by using 4x4 matrix notation to represent the different joints.

The different points and angles are defined like this:



In the all the following code blocks, there will be different types of variables used.

Variables that start with a P represent points,
variables that start with a T represent transform that can be performed on points,
and variables that start with an L are scalar length values.

Note: all angles are measured in radians, and all lengths are measured in meters.

Forward kinematics

The input arguments of the function are the actuated angles of the arm, since these are the only ones measurable by the encoders.

The function returns the projected end position of the end effector.

```
function [x, y, z] = forward_kinematics(theta0, theta1, theta3, theta4)
    %defining dimentions
    LbaseToP1 = 0.065;
    LbaseToP3 = 0.149;

    LP1toP2 = 0.350;
    LP5toP6 = 0.620;
    LP5toP2 = 0.120;
    LP6toP7 = 0.300;

    LP3toP4 = 0.120;
    LP4toP5 = 0.280;
```

The first thing that happens in the forward kinematics is the definition of the dimensions of the different joints, these values are gotten from measuring the physical arm.
The point representing the base rotation (Pbase) is rotated theta0 radians.

```
    %rotation z
    TP0toPbase = [cos(theta0) -sin(theta0) 0 0;
                  sin(theta0)  cos(theta0) 0 0;
                  0            0           1 0;
                  0            0           0 1];
```

From there, all the known transforms to the different points are defined, the transforms are defined with the main idea being, rotation happens along the Z-axis, and translation happens along the X-axis. This is done by first translating shoulder actuated points (P1 and P3) and then rotating them along the x axis.

```
%translation z then rotation x
TPbasetoP1 = [1 0 0  0;
              0 0 -1 0;
              0 1 0  LbaseToP1;
              0 0 0  1];

%rotation z
TPbasetoP1 = TPbasetoP1 * [cos(theta1) -sin(theta1) 0 0;
                           sin(theta1)  cos(theta1) 0 0;
                           0            0           1 0;
                           0            0           0 1];
```

```
%translation z then rotation x
TPbasetoP3 = [1 0  0 0;
              0 0 -1 0;
              0 1  0 LbaseToP3;
              0 0  0 1];

%rotation z
TPbasetoP3 = TPbasetoP3 * [cos(theta4) -sin(theta4) 0 0;
                           sin(theta4)  cos(theta4) 0 0;
                           0            0           1 0;
                           0            0           0 1];
```

The reason that they are rotated twice is because it is easier to keep track of the relative positions and angles involved.

After that, the other currently inferrable transforms are defined.

```
%translation x
TP1toP2 = [1 0 0 LP1toP2;
           0 1 0 0;
           0 0 1 0;
           0 0 0 1];

%translation x then rotation z
TP5toP6 = [cos(theta3) -sin(theta3) 0 LP5toP6;
           sin(theta3)  cos(theta3) 0 0;
           0            0           1 0;
           0            0           0 1];
```

```
    %translation x
    TP6toP7 = [1 0 0 LP6toP7;
               0 1 0 0;
               0 0 1 0;
               0 0 0 1];

    %translation x
    TP3toP4 = [1 0 0 LP3toP4;
               0 1 0 0;
               0 0 1 0;
               0 0 0 1];
```

Some of these transforms don't have rotations, this is because we either don't know the angle rotation yet, or it is an end point, that does not have to be moved and by extension rotated further.

Due to the construction of the arm, some calculations have to be done to find the rest of the unknown angles.
This is done by first defining an initial position, and then some theoretical "planar" positions of the points P2 and P4.

```
    %initial position
    P0 = [1 0 0 0;
          0 1 0 0;
          0 0 1 0;
          0 0 0 1];

    %getting position for paralellagram angles
    P2planar = P0 * TPbasetoP1 * TP1toP2;
    P4planar = P0 * TPbasetoP3 * TP3toP4;
```

The reason they are planar is because they were produces without any base rotation, so they all lay on the y=0 plane.

After that, a small inverse kinematics equation is performed to find the angles theta5 and theta6, which is done using a standard formula.

```
    L_1 = LP4toP5; L_2 = LP5toP2;
    XE = P4planar(1, 4) - P2planar(1, 4);
    YE = P4planar(3, 4) - P2planar(3, 4);
    theta5 = 2*atan((2*L_1*YE + sqrt(- L_1^4 + 2*L_1^2*L_2^2 + 2*L_1^2*XE^2 + 2*L_1^2*YE^2 -
L_2^4 + 2*L_2^2*XE^2 + 2*L_2^2*YE^2 - XE^4 - 2*XE^2*YE^2 - YE^4))/(L_1^2 + 2*L_1*XE - L_2^2 +
```

```
XE^2 + YE^2));
    theta6 = -2*atan(sqrt((- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2)*(L_1^2 + 2*L_1*L_2 +
L_2^2 - XE^2 - YE^2))/(- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2));
    %adding pi-theta4 to make the actual rotation angle
    theta5 = theta5+(pi-theta4);
```

Now that we know all the angles, the rest of the transforms can be defined.

```
    %translation x then rotation z
    TP3toP4 = [cos(theta5) -sin(theta5) 0 LP3toP4;
               sin(theta5)  cos(theta5) 0 0;
               0            0           1 0;
               0            0           0 1];

    %translation x then rotation z
    TP4toP5 = [cos(theta6) -sin(theta6) 0 LP4toP5;
               sin(theta6)  cos(theta6) 0 0;
               0            0           1 0;
               0            0           0 1];
```

And now we have all the transforms, we can calculate all the points.

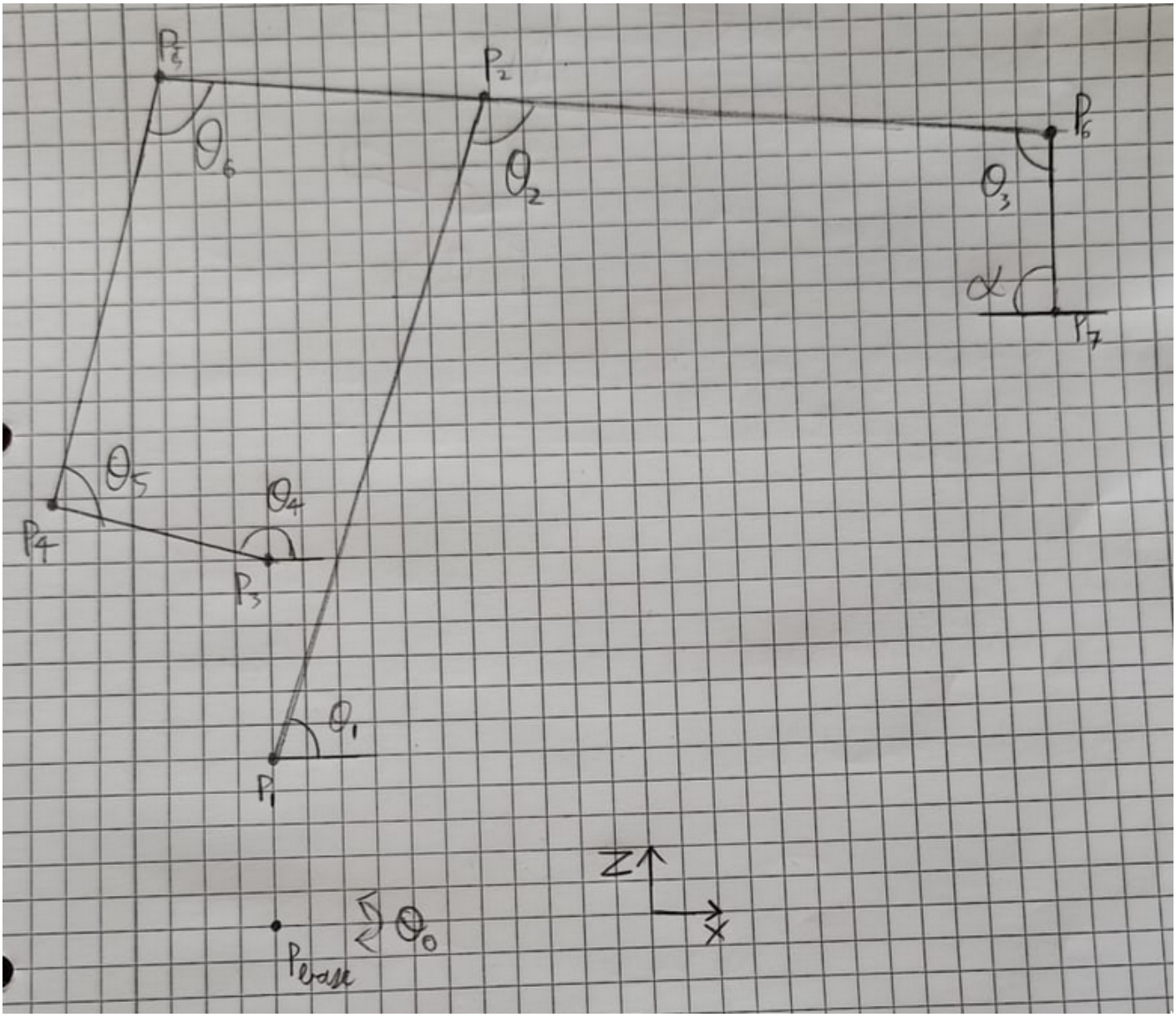
```
    %calculating all needed points
    Pbase = TP0toPbase;
    P3 = Pbase * TPbasetoP3;
    P4 = P3 * TP3toP4;
    P5 = P4 * TP4toP5;
    P6 = P5 * TP5toP6;
    P7 = P6 * TP6toP7;
```

Finally, the final position of the end effector can be read from the final point.

```
    %extracting position
    x = P7(1,4);
    y = P7(2,4);
    z = P7(3,4);
end
```

Inverse kinematics

The inverse kinematics uses the same representation of points and angles to represent the linkages of the arm.



The inputs for this function are the position of the end effector (x, y, z) and a final gripper angle alpha, which is in reference to the ground.
The outputs are the angles of the actuated points as shown above.

The first thing the function does checking a simple error case, as this will not be caught anywhere else otherwise.

```
function [theta0, theta1, theta3, theta4, error] = inverse_kinematics(x, y, z, alpha)
    defaultAngles = [0; pi/2; 0; -pi/2; pi];
    error = 0;

    if x == 0 && y == 0
        theta0 = defaultAngles(1);
```

```
        theta1 = defaultAngles(2);
        %theta2 = defaultAngles(3);
        theta3 = defaultAngles(4);
        theta4 = defaultAngles(5);
        error = 1;
        return;
    end
```

After that the same dimensions as the forward kinematics are defined.

```
    %all the lengths are in meters
    LbaseToP1 = 0.065;
    LbaseToP3 = 0.149;

    LP1toP2 = 0.350;
    LP2toP6 = 0.500;
    %LP5toP6 = 0.620;
    LP5toP2 = 0.120;
    LP6toP7 = 0.300;

    LP3toP4 = 0.120;
    LP4toP5 = 0.280;
```

Since the arm is planar by nature, the angle of the base rotation (theta0) can easily be calculated. A variable called angToBase is also defined, it represents the angle of the end effector to the base of the arm.

```
    if x < 0
        theta0 = atan(y/x) + pi;
    else
        theta0 = atan(y/x);
    end
    angToBase = theta0 - pi;
```

After that, P7 and P6 are defined using the input arguments. The rotation x on P7 is to align rotations with the z-axis and translations with the x-axis. (The i at the end is for inverse.)

```
  %setting end position, rotated towards the base
    P7i = [cos(angToBase) -sin(angToBase) 0 x;
           sin(angToBase)  cos(angToBase) 0 y;
           0               0              1 z;
```

```
       0              0              0 1];

%rotation x
P7i = P7i * [1 0  0 0;
             0 0 -1 0;
             0 1  0 0;
             0 0  0 1];

%rotation z
TP7toP6i = [cos(alpha) -sin(alpha) 0 0;
            sin(alpha)  cos(alpha) 0 0;
            0           0          1 0;
            0           0          0 1];

%translation x
TP7toP6i = TP7toP6i * [1 0 0 LP6toP7;
                       0 1 0  0;
                       0 0 1  0;
                       0 0 0  1];

P6i = P7i * TP7toP6i;
```

P1 is also defined.

```
%translation z
P1i = [1 0 0 0;
       0 1 0 0;
       0 0 1 LbaseToP1;
       0 0 0 1];
```

Now that P1, P6, and P7 are defined, they will be used to calculate theta1, theta2, and theta3.

This is done by first checking if this movement would be possible kinematically, and then actually calculating the angles. This is done with the same standard formula shown in the forward kinematics.

```
%calculating theta1, theta2 and theta3
L_1i = LP1toP2; L_2i = LP2toP6;
dx = P6i(1,4) - P1i(1,4);
dy = P6i(2,4) - P1i(2,4);
XEi = sqrt(dx*dx + dy*dy);
```

```
    YEi = P6i(3,4) - P1i(3,4);

    if (- L_1i^4 + 2*L_1i^2*L_2i^2 + 2*L_1i^2*XEi^2 + 2*L_1i^2*YEi^2 - L_2i^4 + 2*L_2i^2*XEi^2
+ 2*L_2i^2*YEi^2 - XEi^4 - 2*XEi^2*YEi^2 - YEi^4) < 0
        error = 1;
        theta0 = defaultAngles(1);
        theta1 = defaultAngles(2);
        %theta2 = defaultAngles(3);
        theta3 = defaultAngles(4);
        theta4 = defaultAngles(5);
        return;
    end

    theta1 = 2*atan((2*L_1i*YEi + sqrt(- L_1i^4 + 2*L_1i^2*L_2i^2 + 2*L_1i^2*XEi^2 +
2*L_1i^2*YEi^2 - L_2i^4 + 2*L_2i^2*XEi^2 + 2*L_2i^2*YEi^2 - XEi^4 - 2*XEi^2*YEi^2 -
YEi^4))/(L_1i^2 + 2*L_1i*XEi - L_2i^2 + XEi^2 + YEi^2));
    theta2 = -2*atan(sqrt((- L_1i^2 + 2*L_1i*L_2i - L_2i^2 + XEi^2 + YEi^2)*(L_1i^2 +
2*L_1i*L_2i + L_2i^2 - XEi^2 - YEi^2))/(- L_1i^2 + 2*L_1i*L_2i - L_2i^2 + XEi^2 + YEi^2));

    theta3 = 2*pi - alpha -theta1 -theta2;
```

These angles are then used to define transforms to other necessary points. An initial point is also defined.

```
    %translation z then rotation x
    TPbasetoP1 = [1 0 0  0;
                  0 0 -1 0;
                  0 1 0  LbaseToP1;
                  0 0 0  1];

    %rotation z
    TPbasetoP1 = TPbasetoP1 * [cos(theta1) -sin(theta1) 0 0;
                               sin(theta1)  cos(theta1) 0 0;
                               0            0           1 0;
                               0            0           0 1];

    %translation x then rotation z
    TP1toP2 = [cos(theta2) -sin(theta2) 0 LP1toP2;
               sin(theta2)  cos(theta2) 0 0;
               0            0           1 0;
```

```
            0           0           0 1];

    %translation z then rotation x
    TPbasetoP3 = [1 0 0  0;
                  0 0 -1 0;
                  0 1 0  LbaseToP3;
                  0 0 0  1];

    %translation x
    TP2toP5 = [1 0 0 -LP5toP2;
               0 1 0 0;
               0 0 1 0;
               0 0 0 1];

    %initial position
    P0 = [1 0 0 0;
          0 1 0 0;
          0 0 1 0;
          0 0 0 1];
```

These new transforms are now used to interpolate planar versions of P3 and P5.

```
    %getting position for paralellagram angles
    P3planar = P0 * TPbasetoP3;
    P5planar = P0 * TPbasetoP1 * TP1toP2 * TP2toP5;
```

These points are used to calculate the final needed angles, using the same formula as before.

```
    %getting parallelageram angles
    L_1 = LP3toP4; L_2 = LP4toP5;
    XE = P5planar(1, 4) - P3planar(1, 4);
    YE = P5planar(3, 4) - P3planar(3, 4);

    if (- L_1^4 + 2*L_1^2*L_2^2 + 2*L_1^2*XE^2 + 2*L_1^2*YE^2 - L_2^4 + 2*L_2^2*XE^2 +
2*L_2^2*YE^2 - XE^4 - 2*XE^2*YE^2 - YE^4) < 0
        error = 1;
        theta0 = defaultAngles(1);
        theta1 = defaultAngles(2);
        %theta2 = defaultAngles(3);
        theta3 = defaultAngles(4);
```

```
        theta4 = defaultAngles(5);
        return;
    end

    theta4 = 2*atan((2*L_1*YE + sqrt(- L_1^4 + 2*L_1^2*L_2^2 + 2*L_1^2*XE^2 + 2*L_1^2*YE^2 - L_2^4 + 2*L_2^2*XE^2 + 2*L_2^2*YE^2 - XE^4 - 2*XE^2*YE^2 - YE^4))/(L_1^2 + 2*L_1*XE - L_2^2 + XE^2 + YE^2));
```

Note: the angles resulting from the function are equal to the angles defined on the diagram above, which are not equal to the positions the motors have to move to.

Revision #6
Created 2026-04-15 11:07:56 UTC by Rowan Ali
Updated 2026-04-17 13:47:48 UTC by Rowan Ali