ANRO

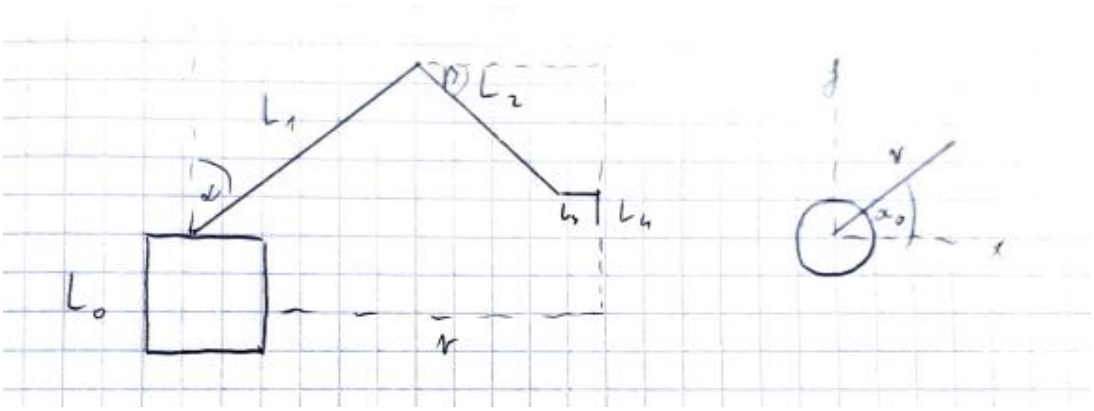
Michał Kwarciński

Kacper Marchlewicz

Praca domowa lab4

1. Odwrotne zadanie kinematyki

Uproszczony schemat robota:



Kąt α_0 można obliczyć następująco:

$$\alpha_0 = \arctan\left(\frac{-x}{y}\right)$$

Ze znanych własności trygonometrycznych wyznaczamy α i β:

$$a = L_1 \sin(\alpha) + L_2 \cos(\beta) = r - L_3$$

$$b = L_1 \cos(\alpha) - L_2 \sin(\beta) = z + L_4 - L_0$$

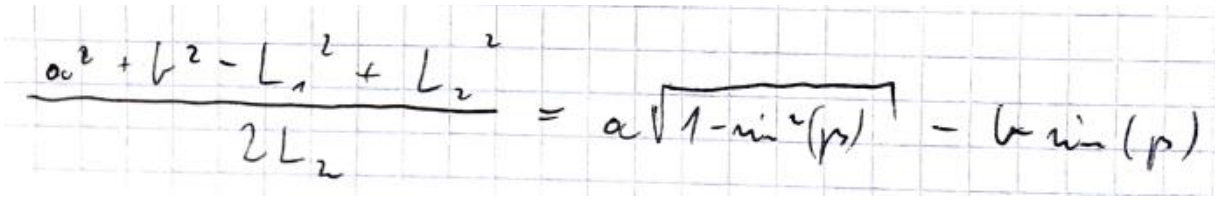
$$r = \sqrt{x^2 + y^2}$$

Na podstawie czego można wyznaczyć sin i cos:

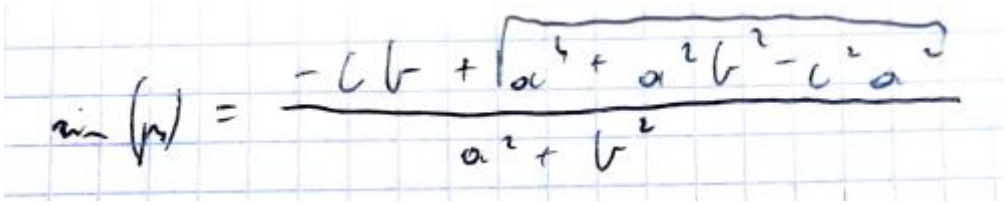
$$\cos(\alpha) = \frac{a - L_1 \sin(\beta)}{L_2}$$

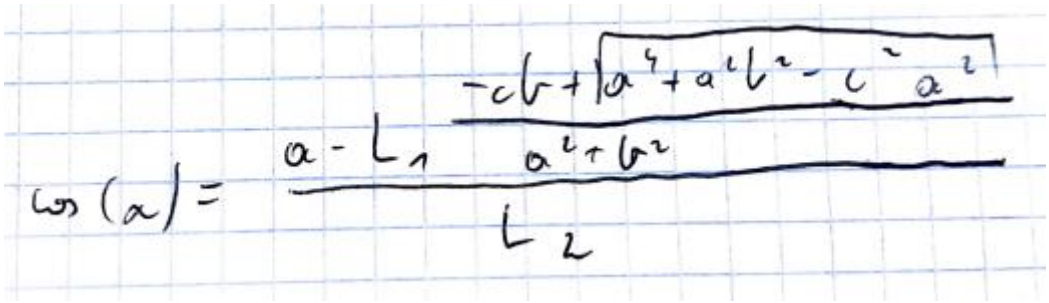
$$\sin(\alpha) = \frac{b + L_1 \cos(\beta)}{L_2}$$

Podnosząc równania do kwadratu wyznaczamy:



Podstawiając za lewą stronę „c” w celu prostszego zapisu:





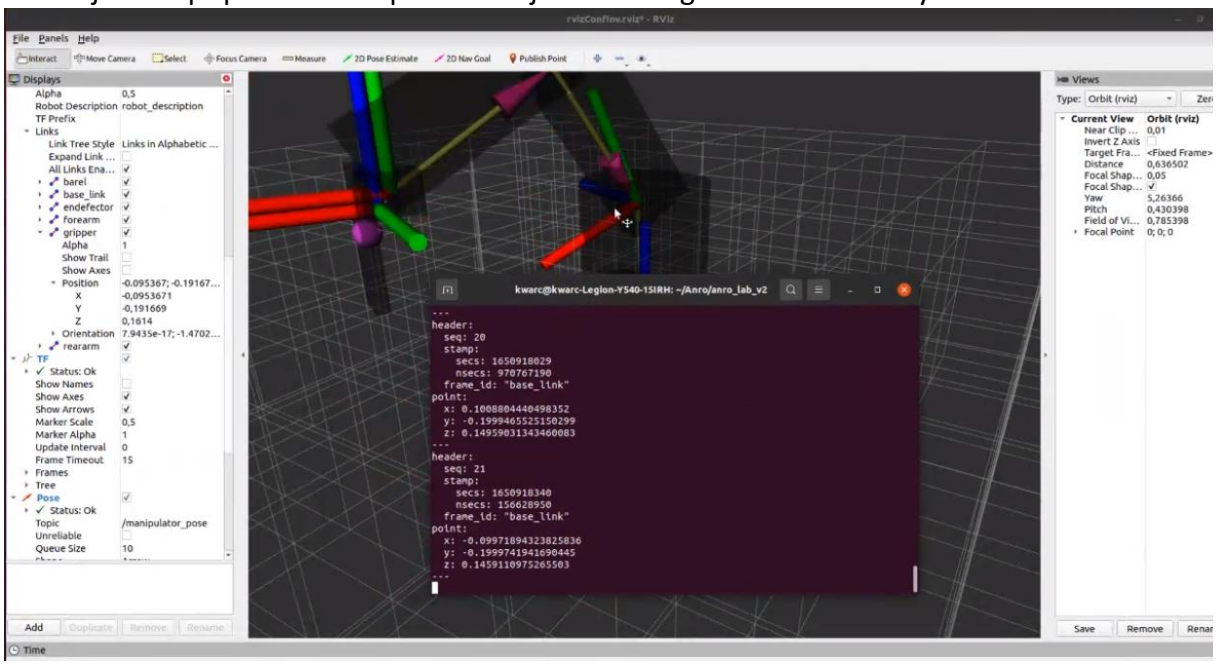
Znając ich wartości będziemy w stanie obliczyć arccos i arcsin.

Kod pliku KDL_Inv

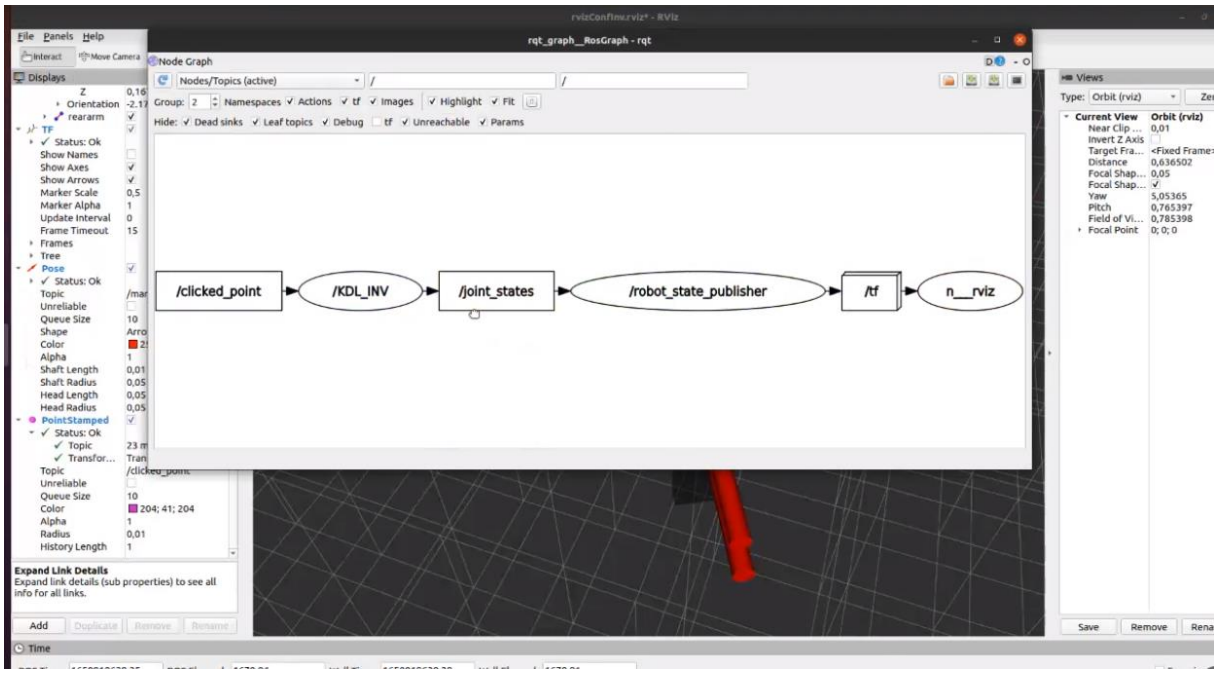
```
#include <ros/ros.h>
#include "sensor_msgs/JointState.h"
#include <stdio.h>
#include "std_msgs/String.h"
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/PointStamped.h>
#include <cmath>
#include <iostream>
#include <cstdlib>

double x=0;
double y=0;
double z=0.025;

void callback_position(const geometry_msgs::PointStamped::ConstPtr& msg)
{
    x = msg->point.x;
    y = msg->point.y;
    z = msg->point.z;
}

int main( int argc, char** argv)
{
ros::init(argc, argv, "KDL_INV");
ros::NodeHandle nh;
ros::Publisher publisher =
nh.advertise<sensor_msgs::JointState>("/joint_states", 1000);
```

```
ros::Subscriber subscriber = nh.subscribe("/clicked_point", 1000,
callback_position);

ros::Rate rate(20);

sensor_msgs::JointState msg;
msg.header.frame_id = "";
msg.position.resize(5);
msg.name.resize(5);
msg.name[0] = "joint_1";
msg.name[1] = "joint_2";
msg.name[2] = "joint_3";
msg.name[3] = "joint_4";
msg.name[4] = "joint_5";

while(ros::ok())
{
    double a1;
    double a2;
    double a3;
    double a4;
    double a5;

    double r = sqrt(pow(x,2) + pow(y,2));
    double a = r -0.01;
    double b = z +0.01 - 0.11;
    double c = (pow(a,2) + pow(b,2) - 0.024025 + 0.027889)/(0.334);

    if(y!=0)
    {
        a1 = atan(-x/y);
    }
    else
    {
        a1 = 0;
    }
    double root = sqrt(pow(a,4)+pow(a,2)*pow(b,2)-pow(c,2)*pow(a,2));
    double tmp = pow(a,2)+pow(b,2);
    a2 =-acos((b+0.1469*(-c*b+root)/tmp)/0.1524)+0.178;
    a3 =-asin((-c*b+root)/tmp)-a2+1.067;
    a4 = -a3-a2;
    a5 = 0;

    msg.header.stamp = ros::Time::now();
    msg.position[0]=a1;
    msg.position[1]=a2;
    msg.position[2]=a3;
    msg.position[3]=a4;
    msg.position[4]=a5;
    publisher.publish(msg);

    rate.sleep();
    ros::spinOnce();
 }
 return 0;
}
```

Poniżej widać poprawność implementacji odwrotnego zdania kinematyki:



Schemat struktury systemu:



Stworzony przez nas KDL_INV odbiera pozycję od /clicked_point i przekazuje obliczone wartości kątów na temat /joint_states, które trafiają do n__rviz.