### **−**Table of Contents

## Introduction

In this assignment we learned forward and inverse kinematics on the UR10 robot. Forward kinematics is the calculation of en point with given joint values. Inverse kinematics is the finding this values for a given point. We used Orocos Kinematics and Dynamics Library(KDL). Using the library we first defined UR10 robot with its 6 joints. Then we used built in library codes to calculate Forward and Inverse kinematic values. After that we calculated error by comparing them with their ideal values.

Formun Üstü

Formun Altı

## UR10 DH and frame construction

To construct frame first I subscribed the join positions. From there I get this :

Base Position :[ 0.241427 , -2.05357e-07 , 0.04465 ]

joint positions are relative to first joint(base)

Joint0 Position :[ 0 , 0 , 0 ]

Joint1 Position :[ -0.0926501 , -4.23356e-08 , 0.08335 ]

Joint2 Position :[ -0.112263 , 0.000568758 , 0.695438 ]

Joint3 Position :[ -0.1058 , -6.32342e-05 , 1.26764 ]

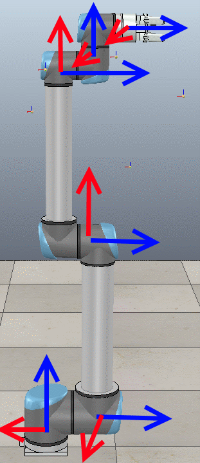
Joint4 Position :[ -0.164229 , -6.33453e-05 , 1.32492 ]

Joint5 Position :[ -0.221548 , -6.3571e-05 , 1.3833 ]

End Effector Position :[ -0.332061 , -2.44681e-05 , 1.38318 ]

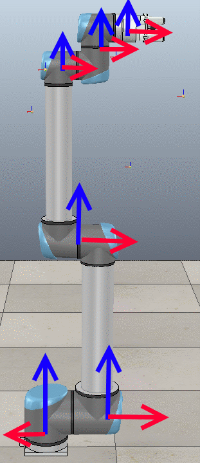
Using the values with below axises I constructed the dh paramaters:   
**To make things easier I asumed all joints have 0 y axis (ideally its ~e-05 which is really close)**

| DH joint | teta | **d** | **a** | **alfa** |
| --- | --- | --- | --- | --- |
| **1** | PI/2 | 0.08335 | 0 | 0 |
| **2** | -PI/2 | -(-0.112263–0.0926501) | 0.695438-0.08335 | 0 |
| **3** | 0 | -(-0.1058–0.112263) | 1.26764 - 0.695438 | 0 |
| **4** | PI/2 | -(-0.164229–0.1058) | 0 | PI/2 |
| **5** | 0 | 1.3833 -1.32492 | 0 | -PI/2 |
| **6** | 0 | -(-0.332061 –0.221548) | 0 | 0 |

  
red arrow = x / blue arrow = z

Dh paramaters:

| DH joint | teta | **d** | **a** | **alfa** |
| --- | --- | --- | --- | --- |
| **1** | PI/2 | 0.08335 | 0 | 0 |
| **2** | -PI/2 | 0.0196169 | 0.612088 | 0 |
| **3** | 0 | -0.006463 | 0.572206 | 0 |
| **4** | PI/2 | 0.058429 | 0 | PI/2 |
| **5** | 0 | 0.05838 | 0 | -PI/2 |
| **6** | 0 | -0.1883419 | 0 | 0 |

In dh joints are always turning in their Z axis. This is not a case for the KDL frame.   
  
red arrow = x / blue arrow = z

KDL chain:

KDL::Chain chain;

//DO You should write the correct chain for UR10

KDL::Segment s0 = KDL::Segment(KDL::Joint(KDL::Joint::RotZ), //z

KDL::Frame(KDL::Rotation::RPY(0.0, 0.0 ,PI),

KDL::Vector(-0.0926501 , -4.23356e-08 , 0.08335) )

);

KDL::Segment s1 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.0926501 - (-0.112263) ,-4.23356e-08 - (0.000568758) , 0.695438 - 0.08335) )

);

KDL::Segment s2 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.112263 - (-0.1058),0.000568758 - (-6.32342e-05),1.26764-0.695438 ) )

);

KDL::Segment s3 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.1058 - (-0.164229),-6.32342e-05 -(-6.33453e-05),1.32492-1.26764) )

);

KDL::Segment s4 = KDL::Segment(KDL::Joint(KDL::Joint::RotZ), //z

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.164229 - (-0.221548), -6.33453e-05 - (-6.3571e-05),1.3833-1.32492) )

);

KDL::Segment s5 = KDL::Segment(KDL::Joint(KDL::Joint::RotX), //-x

KDL::Frame(KDL::Rotation::RPY(0.0,0.0,0.0),

KDL::Vector(-0.221548 - (-0.332061) ,-6.3571e-05 - (-2.44681e-05) ,1.38318-1.3833) )

);

chain.addSegment(s0);

chain.addSegment(s1);

chain.addSegment(s2);

chain.addSegment(s3);

chain.addSegment(s4);

chain.addSegment(s5);

return chain;

Formun Üstü

Formun Altı

## Forward Kinamatics

Read file:

std::ifstream infile ;

infile.open("/home/alpsark/ros-ws/src/ur10\_kinematics/src/angles");

if(infile){

infile >> jointValues[0] >> jointValues[1] >>jointValues[2] >> jointValues[3] >> jointValues[4] >> jointValues[5] ;

}else{printf("%s \n","Error: could not open file :(");}

inside the while loop:

unsigned int nj = chain.getNrOfJoints();

KDL::JntArray jointpositions = KDL::JntArray(nj);

for(int index = 0; index<nj ; index++){

jointpositions(index) = jointValues[index] \* PI / 180 ;}

// Create the frame that will contain the results

KDL::Frame cartpos;

// Create solver based on kinematic chain

KDL::ChainFkSolverPos\_recursive fksolver = KDL::ChainFkSolverPos\_recursive(chain);

// Calculate forward position kinematics

bool kinematics\_status;

kinematics\_status = fksolver.JntToCart(jointpositions,cartpos);

if(kinematics\_status>=0){

}else{

printf("%s \n","Error: could not calculate forward kinematics :(");

}

/\*

\* DO: Publish the found joint angels to topic.

\*/

std\_msgs::Float32 jointMessage;

jointMessage.data = jointpositions(0);

joints[0].publish(jointMessage);

jointMessage.data = jointpositions(1);

joints[1].publish(jointMessage);

jointMessage.data = jointpositions(2);

joints[2].publish(jointMessage);

jointMessage.data = jointpositions(3);

joints[3].publish(jointMessage);

jointMessage.data = jointpositions(4);

joints[4].publish(jointMessage);

jointMessage.data = jointpositions(5);

joints[5].publish(jointMessage);

/\*

\* DO: Fill the calculatedLocation variable for error calculation.

\*/

calculatedPosition.position.x = cartpos.p[0];

calculatedPosition.position.y = cartpos.p[1];

calculatedPosition.position.z = cartpos.p[2];

Results:  
<https://youtu.be/h6lvk7Zo0p4>

| Data | joint1 | **joint2** | **joint3** | **joint4** | **joint5** | **joint6** | **error** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 0 | 0 | -90 | 90 | -180 | 0 | 0.103753 |
| **2** | -30 | 45 | -45 | 0 | 0 | 0 | 0.172084 |
| **3** | 0 | 0 | 0 | 0 | 90 | -90 | 0.0177558 |
| **4** | 0 | -90 | 90 | 0 | 0 | 90 | 0.148911 |
| **5** | 40 | 0 | -84 | 90 | 45 | 0 | 0.119892 |

Formun Üstü

Formun Altı

## Inverse Kinamatics

explanation:

I used KDL::ChainIkSolverPos\_LMA object, this solves the inverse kinematic using Levenberg–Marquardt algorithm. Algorithm uses iterative prosedure to implement gradient descent. With each iteration it converges to local minumum, which in our case is target point.

code:

//create iksolver

KDL::ChainIkSolverPos\_LMA iksolver = KDL::ChainIkSolverPos\_LMA(chain,L );

// Create joint arrays

unsigned int nj = chain.getNrOfJoints();

KDL::JntArray initialjointpositions = KDL::JntArray(nj);

KDL::JntArray jointpositions = KDL::JntArray(nj);

int ret = iksolver.CartToJnt(initialjointpositions, goalEndEffectorPosition, jointpositions);

std::cout << "Joint angles for target " << currentTarget << " : "<< jointpositions(0) << " " << jointpositions(1) << " " <<jointpositions(2) << " " <<jointpositions(3) << " " <<jointpositions(4) << " " <<jointpositions(5) << " " <<std::endl;

/\*

\* DO: Publish the found joint angles.

\*/

if(ret == 0) {

std\_msgs::Float32 jointMessage;

jointMessage.data = jointpositions(0);

joints[0].publish(jointMessage);

jointMessage.data = jointpositions(1);

joints[1].publish(jointMessage);

jointMessage.data = jointpositions(2);

joints[2].publish(jointMessage);

jointMessage.data = jointpositions(3);

joints[3].publish(jointMessage);

jointMessage.data = jointpositions(4);

joints[4].publish(jointMessage);

jointMessage.data = jointpositions(5);

joints[5].publish(jointMessage);}else{ printf("%s \n","Error: could not calculate inverse kinematics :("); }

}

Results:  
<https://youtu.be/0Rgah8KTy0E>

|  |  |
| --- | --- |
| target | error |
| **0** | 0.140234 |
| **1** | 0.170797 |
| **2** | 0.130535 |
| **3** | 0.210088 |
| **4** | 0.118855 |

Formun Üstü

Formun Altı

## Bonus part

code:   
I added couple of validity variables.   
**is\_cup\_taken** true if robot has passed cupPositions.   
**cup\_gripped** true if cup is gripped by robot .   
**gone\_midway** true if robot has passed midway between cupPositions and cupTargetPositions.

I fine tuned the variables for smooth motion. I alse added an extra target ponit between cupPositions and cupTargetPositions(Otherwise robot crashes the rack).

I combined the first two kinematics to create the code.

KDL::ChainFkSolverPos\_recursive forwardKinematicSolver = KDL::ChainFkSolverPos\_recursive(chain);

KDL::Frame goalEndEffectorPosition;

bool isCurrentSetted = false;

std\_msgs::Float32 jointMessage;

jointMessage.data = 5.5;

if(is\_cup\_taken) {gripper.publish(jointMessage); gripper\_count++ ;

if(gripper\_count > 10) {goalPositions = cupTargetPositions;cup\_gripped = true;}

}else { goalPositions = cupPositions;

}

if(0 < goalPositions.poses.size()) {

calculatedPosition = goalPositions.poses.at(0);

isCurrentSetted = true;

}

if(isCurrentSetted) { if(cup\_gripped){

if(gone\_midway){ goalEndEffectorPosition = KDL::Frame(

KDL::Rotation::RPY(0,0,PI),

KDL::Vector(calculatedPosition.position.x,calculatedPosition.position.y,calculatedPosition.position.z));}

else{goalEndEffectorPosition = KDL::Frame(

KDL::Rotation::RPY(0,0,PI),

KDL::Vector(calculatedPosition.position.x/2,calculatedPosition.position.y,calculatedPosition.position.z));}

}else{ goalEndEffectorPosition = KDL::Frame(

KDL::Rotation::RPY(0,0,0),

KDL::Vector( calculatedPosition.position.x-0.08,calculatedPosition.position.y,calculatedPosition.position.z)

);}

// You can use this matrix for ChainIkSolverPos\_LMA

Eigen::Matrix<double,6,1> L;

L(0)=1;L(1)=1;L(2)=1;

L(3)=0.01;L(4)=0.01;L(5)=0.01;

//create iksolver

KDL::ChainIkSolverPos\_LMA iksolver = KDL::ChainIkSolverPos\_LMA(chain,L );

// Create joint arrays

unsigned int nj = chain.getNrOfJoints();

KDL::JntArray initialjointpositions = KDL::JntArray(nj);

KDL::JntArray jointpositions = KDL::JntArray(nj);

int ret = iksolver.CartToJnt(initialjointpositions, goalEndEffectorPosition, jointpositions);

/\*

\* DO: Publish the found joint angles.

\*/

if(ret == 0) {

jointMessage.data = jointpositions(0);

joints[0].publish(jointMessage);

jointMessage.data = jointpositions(1);

joints[1].publish(jointMessage);

jointMessage.data = jointpositions(2);

joints[2].publish(jointMessage);

jointMessage.data = jointpositions(3);

joints[3].publish(jointMessage);

jointMessage.data = jointpositions(4);

joints[4].publish(jointMessage);

jointMessage.data = jointpositions(5);

joints[5].publish(jointMessage);}else{ std::cout << ret ; printf("%s \n","Error: could not calculate inverse kinematics :("); }

}}

Results:  
<https://youtu.be/5ALUBEdDziU>

My error was 0.212212.

Formun Üstü

Formun Altı

## Conclusion

I learned to use KLD library and how to use its inverse and forward kinematics. So, by defining a concrete robot chain I can now control most of the non-mobile robots.

Formun Üstü

Formun Altı