Group Assignment – Part 3

Shiva Surya Lolla
Chaitanya Sriram Gaddipati
Ethan Wilke

The Code - Velocity Kinematics

The below code implements two services. The first service obtains joint velocities from end effector velocity. The second service obtains end effector velocity from joint velocities. Below is the documented code of the same.

```
#include <chrono>
#include <functional>
#include <iostream>
#include <cstdlib>
#include <math.h>
#include <memory>
#include <string>
#include <vector>
#include <Eigen/Dense>
#include <unsupported/Eigen/FFT>
#include "sensor_msgs/msg/joint_state.hpp"
#include "tutorial_interfaces/srv/end_to_joint.hpp"
#include "tutorial_interfaces/srv/joint_to_end.hpp"
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/float64_multi_array.hpp"
#include <iterator>
#include <optional>
#include <ratio>
#include <thread>
#include <time.h>
#include <utility>

using std::placeholders::_1;
using namespace std::chrono_literals;

class VelocityKinematics : public rclcpp::Node
{
public:
 VelocityKinematics() : Node("vel_kinematics")
 {
   //Service for obtaining joint velocities from end effector velocity
   service_1 =
this->create_service<tutorial_interfaces::srv::EndToJoint>("end_to_joint",
std::bind(&VelocityKinematics::velref, this, _1));
   //Service for obtaining end effector velocity from joint velocities
   service_2 =
this->create_service<tutorial_interfaces::srv::JointToEnd>("joint_to_end",
std::bind(&VelocityKinematics::jref, this, _1));
 }

private:
 mutable double evx_ref;
 mutable double evy_ref;
 mutable double evz_ref;
 mutable double ewx_ref;
 mutable double ewy_ref;
 mutable double ewz_ref;
 mutable double j1v_ref;
 mutable double j2v_ref;
 mutable double j3v_ref;

 void velref(const
std::shared_ptr<tutorial_interfaces::srv::EndToJoint::Request>
                 request)
 {
   //Taking end effector velocity from request
   evx_ref = request->vx_ref;
   evy_ref = request->vy_ref;
   evz_ref = request->vz_ref;
   ewx_ref = request->omx_ref;
   ewy_ref = request->omy_ref;
   ewz_ref = request->omz_ref;
   RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Incoming request\nvx_ref:
%f" " vy_ref: %f" " vz_ref: %f", request->vx_ref, request->vy_ref,
request->vz_ref);
   //Subscribing to current joint states for getting joint angle values
for computing Jacobian matrix
   subscriber_1 =
this->create_subscription<sensor_msgs::msg::JointState>("/joint_states",
10, std::bind(&VelocityKinematics::topic_callback1, this, _1));
```

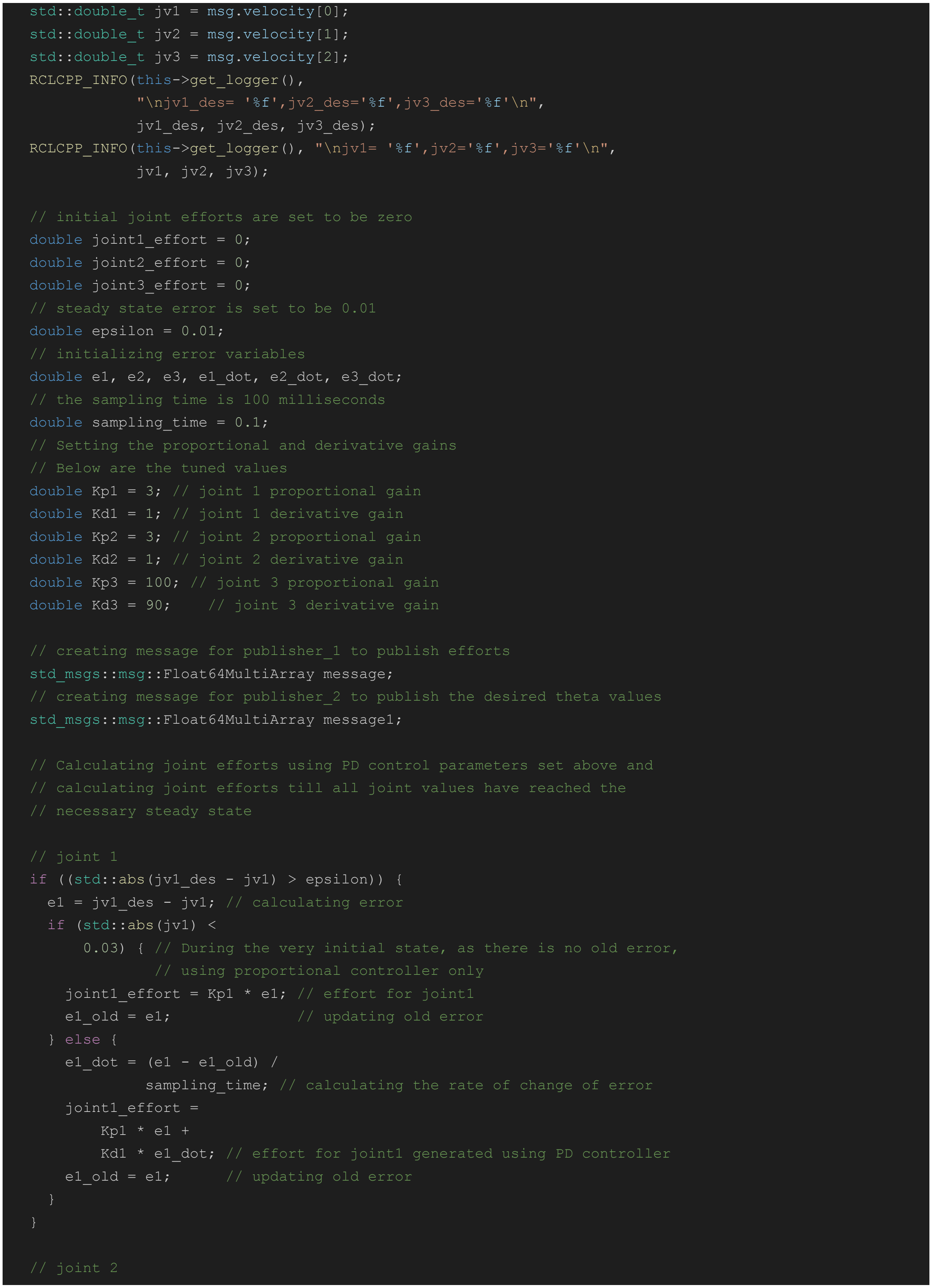
```
}

void topic_callback1(const sensor_msgs::msg::JointState &msg) const {
  //Extracting current position information
  std::double_t q1 = msg.position[0];
  std::double_t q2 = msg.position[1];
  std::double_t q3 = msg.position[2];
  //Computing pseudo inverse of Jacobian matrix
  double sig1 = sin(q1)*sin(q2);
  double sig2 = cos(q2)*sin(q1);
  double sig3 = cos(q1)*sin(q2);
  double sig4 = cos(q1)*cos(q2);
  double sq1 = sin(q1);
  double cq1 = cos(q1);
  Eigen::MatrixXd J(6,3);
  J(0,0) = -sq1-sig3-sig2;
  J(0,1) = -sig3-sig2;
  J(0,2) = 0;
  J(1,0) = cq1+sig4-sig1;
  J(1,1) = sig4-sig1;
  J(1,2) = 0;
  J(2,0) = 0;
  J(2,1) = 0;
  J(2,2) = -1;
  J(3,0) = 0;
  J(3,1) = 0;
  J(3,2) = 0;
  J(4,0) = 0;
  J(4,1) = 0;
  J(4,2) = 0;
  J(5,0) = 1;
  J(5,1) = 1;
  J(5,2) = 0;
  Eigen::MatrixXd pinvJ =
J.completeOrthogonalDecomposition().pseudoInverse();

  Eigen::MatrixXd vmat(6,1);
  vmat(0,0) = evx_ref;
  vmat(1,0) = evy_ref;
```

```
   vmat(2,0) = evz_ref;
   vmat(3,0) = ewx_ref;
   vmat(4,0) = ewy_ref;
   vmat(5,0) = ewz_ref;

   //Computing joint velocities from pseudo inverse of Jacobian matrix and
end effector velocity
   Eigen::MatrixXd jmat = pinvJ*vmat;
   j1v_ref = jmat(0,0);
   j2v_ref = jmat(1,0);
   j3v_ref = jmat(2,0);
   //Sending the obtained joint velocities as response
   tutorial_interfaces::srv::EndToJoint::Response response;
   response.joint1vel_ref = j1v_ref;
   response.joint2vel_ref = j2v_ref;
   response.joint3vel_ref = j3v_ref;
   RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "sending back response:\n
[%f, %f, %f]", response.joint1vel_ref, response.joint2vel_ref,
response.joint3vel_ref);
 }
 rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr
subscriber_1;
 rclcpp::Service<tutorial_interfaces::srv::EndToJoint>::SharedPtr
service_1;

private:
 mutable double j1_refv;
 mutable double j2_refv;
 mutable double j3_refv;
 mutable double v_refx;
 mutable double v_refy;
 mutable double v_refz;

 void jref(const
std::shared_ptr<tutorial_interfaces::srv::JointToEnd::Request>
                 request1)
 {
   //Taking joint velocities from request
   j1_refv = request1->j1vel_ref;
   j2_refv = request1->j2vel_ref;
   j3_refv = request1->j3vel_ref;
   RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Incoming request\nj1vel_ref:
%f" " j2vel_ref: %f" " j3vel_ref: %f", request1->j1vel_ref,
request1->j2vel_ref, request1->j3vel_ref);
   //Subscribing to current joint states for getting joint angle values
for computing Jacobian matrix
   subscriber_2 =
this->create_subscription<sensor_msgs::msg::JointState>("/joint_states",
10, std::bind(&VelocityKinematics::topic_callback2, this, _1));
 }

 void topic_callback2(const sensor_msgs::msg::JointState &msg2) const {
   //Extracting current position information
   std::double_t q1 = msg2.position[0];
   std::double_t q2 = msg2.position[1];
   std::double_t q3 = msg2.position[2];
   //Computing Jacobian matrix
   double sig1 = sin(q1)*sin(q2);
   double sig2 = cos(q2)*sin(q1);
   double sig3 = cos(q1)*sin(q2);
   double sig4 = cos(q1)*cos(q2);
   double sq1 = sin(q1);
   double cq1 = cos(q1);
   Eigen::MatrixXd J(6,3);
   J(0,0) = -sq1-sig3-sig2;
   J(0,1) = -sig3-sig2;
   J(0,2) = 0;
   J(1,0) = cq1+sig4-sig1;
   J(1,1) = sig4-sig1;
   J(1,2) = 0;
   J(2,0) = 0;
   J(2,1) = 0;
   J(2,2) = -1;
   J(3,0) = 0;
   J(3,1) = 0;
   J(3,2) = 0;
   J(4,0) = 0;
   J(4,1) = 0;
   J(4,2) = 0;
   J(5,0) = 1;
   J(5,1) = 1;
   J(5,2) = 0;

   Eigen::MatrixXd jvmat(3,1);
   jvmat(0,0) = j1_refv;
   jvmat(1,0) = j2_refv;
   jvmat(2,0) = j3_refv;

   //Computing end effector velocity from Jacobian matrix and joint velocities
   Eigen::MatrixXd vemat = J*jvmat;
   v_refx = vemat(0,0);
   v_refy = vemat(1,0);
   v_refz = vemat(2,0);
   //Sending the obtained end effector joint velocities as response
   tutorial_interfaces::srv::JointToEnd::Response response1;
   response1.vref_x = v_refx;
   response1.vref_y = v_refy;
   response1.vref_z = v_refz;
   RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "sending back response:\n [%f, %f, %f]", response1.vref_x, response1.vref_y, response1.vref_z);
 }
 rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr subscriber_2;
 rclcpp::Service<tutorial_interfaces::srv::JointToEnd>::SharedPtr service_2;
};

int main(int argc, char *argv[]) {
 rclcpp::init(argc, argv);
 rclcpp::spin(std::make_shared<VelocityKinematics>());
 rclcpp::shutdown();
 return 0;
}
```

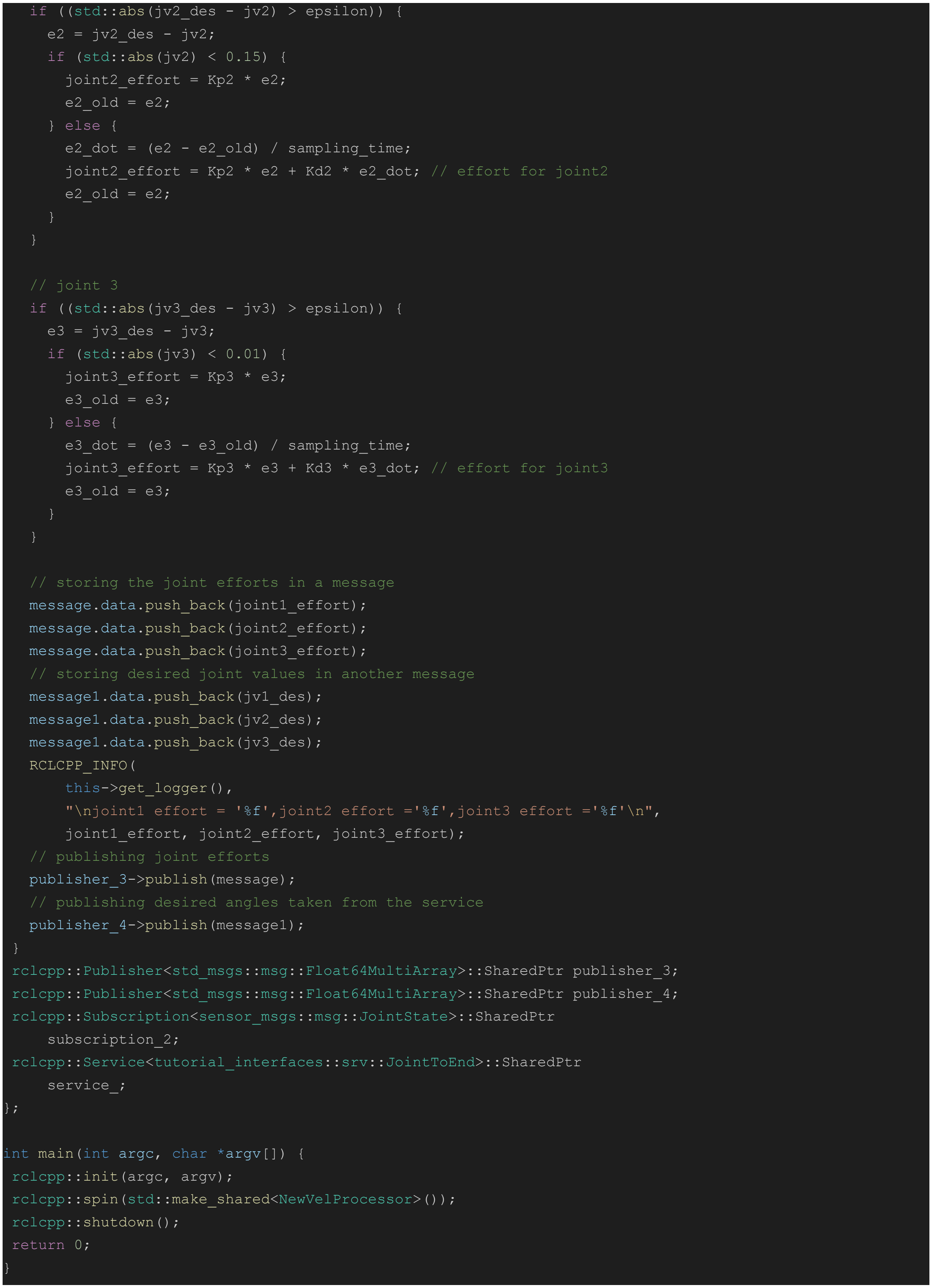
The Code - Velocity Controller:

The “Velocity control file” consists of a subscriber file that reads and organizes the recorded current velocities of the joints and then uses them for the control of joint velocities. Below is its code. (Please check code comments below for documentation)

```
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include "std_msgs/msg/float64_multi_array.hpp"
#include "tutorial_interfaces/srv/joint_to_end.hpp"
#include <chrono>
#include <cstdlib>
#include <functional>
#include <iostream>
#include <iterator>
#include <memory>
#include <optional>
#include <ratio>
#include <thread>
#include <time.h>
#include <utility>
#include <vector>
using std::placeholders::_1;
using namespace std::chrono_literals;
using namespace std::chrono;

class NewVelProcessor : public rclcpp::Node {
public:
 NewVelProcessor() : Node("velocity_control") {
   // create a service to take desired joint values
   service_ = this->create_service<tutorial_interfaces::srv::JointToEnd>(
       "joint_to_end", std::bind(&NewVelProcessor::jointvelref, this, _1));
 }

private:
 // initializing the old error and desired joint values for modification inside
 // the callback function
 mutable double e1_old;
 mutable double e2_old;
 mutable double e3_old;
 mutable double jv1_des;
 mutable double jv2_des;
 mutable double jv3_des;

 void jointvelref(
     const std::shared_ptr<tutorial_interfaces::srv::JointToEnd::Request> request) {
   // taking desired joint velocity values from the client request from the terminal
   jv1_des = request->j1vel_ref;
   jv2_des = request->j2vel_ref;
   jv3_des = request->j3vel_ref;
   // subscribing to actual joint velocities
   subscription_2 = this->create_subscription<sensor_msgs::msg::JointState>("/joint_states", 10,
std::bind(&NewVelProcessor::topic2_callback, this, _1));
   publisher_3 = this->create_publisher<std_msgs::msg::Float64MultiArray>(
       "/forward_effort_controller/commands", 10); // publishing joint efforts
   publisher_4 = this->create_publisher<std_msgs::msg::Float64MultiArray>(
       "/joint_vel_des", 10); // publishing desired joint values for plotting
 }

 void topic2_callback(const sensor_msgs::msg::JointState &msg) const {
   // extracting the actual joint values
```



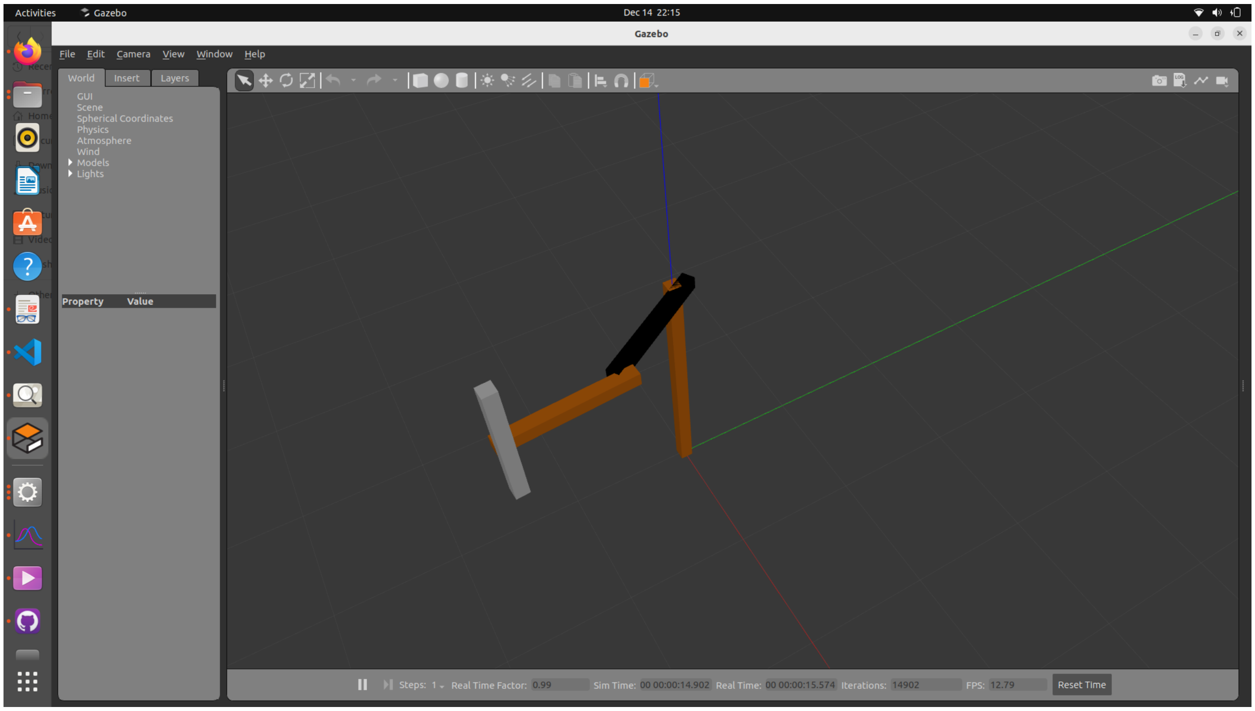
```cpp
std::double_t jv1 = msg.velocity[0];
std::double_t jv2 = msg.velocity[1];
std::double_t jv3 = msg.velocity[2];
RCLCPP_INFO(this->get_logger(),
            "\njv1_des= '%f',jv2_des='%f',jv3_des='%f'\n",
            jv1_des, jv2_des, jv3_des);
RCLCPP_INFO(this->get_logger(), "\njv1= '%f',jv2='%f',jv3='%f'\n",
            jv1, jv2, jv3);

// initial joint efforts are set to be zero
double joint1_effort = 0;
double joint2_effort = 0;
double joint3_effort = 0;
// steady state error is set to be 0.01
double epsilon = 0.01;
// initializing error variables
double e1, e2, e3, e1_dot, e2_dot, e3_dot;
// the sampling time is 100 milliseconds
double sampling_time = 0.1;
// Setting the proportional and derivative gains
// Below are the tuned values
double Kp1 = 3; // joint 1 proportional gain
double Kd1 = 1; // joint 1 derivative gain
double Kp2 = 3; // joint 2 proportional gain
double Kd2 = 1; // joint 2 derivative gain
double Kp3 = 100; // joint 3 proportional gain
double Kd3 = 90;    // joint 3 derivative gain

// creating message for publisher_1 to publish efforts
std_msgs::msg::Float64MultiArray message;
// creating message for publisher_2 to publish the desired theta values
std_msgs::msg::Float64MultiArray message1;

// Calculating joint efforts using PD control parameters set above and
// calculating joint efforts till all joint values have reached the
// necessary steady state

// joint 1
if ((std::abs(jv1_des - jv1) > epsilon)) {
  e1 = jv1_des - jv1; // calculating error
  if (std::abs(jv1) <
      0.03) { // During the very initial state, as there is no old error,
              // using proportional controller only
    joint1_effort = Kp1 * e1; // effort for joint1
    e1_old = e1;              // updating old error
  } else {
    e1_dot = (e1 - e1_old) /
             sampling_time; // calculating the rate of change of error
    joint1_effort =
        Kp1 * e1 +
        Kd1 * e1_dot; // effort for joint1 generated using PD controller
    e1_old = e1;      // updating old error
  }
}

// joint 2
```



```
   if ((std::abs(jv2_des - jv2) > epsilon)) {
     e2 = jv2_des - jv2;
     if (std::abs(jv2) < 0.15) {
       joint2_effort = Kp2 * e2;
       e2_old = e2;
     } else {
       e2_dot = (e2 - e2_old) / sampling_time;
       joint2_effort = Kp2 * e2 + Kd2 * e2_dot; // effort for joint2
       e2_old = e2;
     }
   }

   // joint 3
   if ((std::abs(jv3_des - jv3) > epsilon)) {
     e3 = jv3_des - jv3;
     if (std::abs(jv3) < 0.01) {
       joint3_effort = Kp3 * e3;
       e3_old = e3;
     } else {
       e3_dot = (e3 - e3_old) / sampling_time;
       joint3_effort = Kp3 * e3 + Kd3 * e3_dot; // effort for joint3
       e3_old = e3;
     }
   }

   // storing the joint efforts in a message
   message.data.push_back(joint1_effort);
   message.data.push_back(joint2_effort);
   message.data.push_back(joint3_effort);
   // storing desired joint values in another message
   message1.data.push_back(jv1_des);
   message1.data.push_back(jv2_des);
   message1.data.push_back(jv3_des);
   RCLCPP_INFO(
       this->get_logger(),
       "\njoint1 effort = '%f',joint2 effort ='%f',joint3 effort ='%f'\n",
       joint1_effort, joint2_effort, joint3_effort);
   // publishing joint efforts
   publisher_3->publish(message);
   // publishing desired angles taken from the service
   publisher_4->publish(message1);
 }
 rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr publisher_3;
 rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr publisher_4;
 rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr
     subscription_2;
 rclcpp::Service<tutorial_interfaces::srv::JointToEnd>::SharedPtr
     service_;
};

int main(int argc, char *argv[]) {
 rclcpp::init(argc, argv);
 rclcpp::spin(std::make_shared<NewVelProcessor>());
 rclcpp::shutdown();
 return 0;
}
```

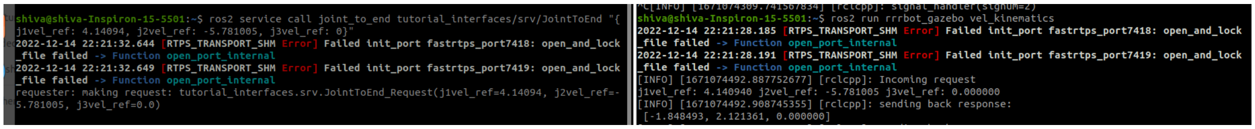
The Results:

Initial joints position is [1,0.5,-0.2] and is achieved using the position_publisher.cpp file already provided



Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:00:14.902 Real Time: 00 00:00:15.574 Iterations: 14902 FPS: 12.79

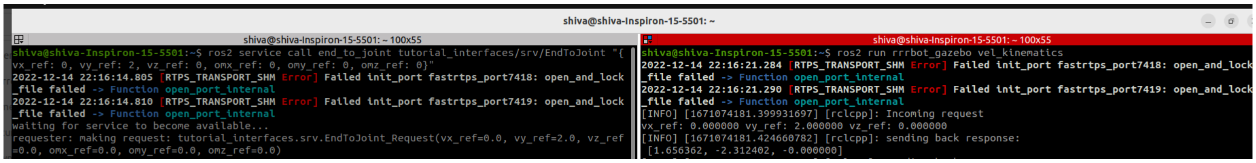
Initial Pos



```
shiva@shiva-Inspiron-15-5501:~$ ros2 service call joint_to_end tutorial_interfaces/srv/JointToEnd "{
j1vel_ref: 4.14094, j2vel_ref: -5.781005, j3vel_ref: 0}"
2022-12-14 22:21:32.644 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:21:32.649 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
requester: making request: tutorial_interfaces.srv.JointToEnd_Request(j1vel_ref=4.14094, j2vel_ref=-
5.781005, j3vel_ref=0.0)
```

```
^C[INFO] [1671074309.741567834] [rclcpp]: signal_handler(signum=2)
shiva@shiva-Inspiron-15-5501:~$ ros2 run rrrbot_gazebo vel_kinematics
2022-12-14 22:21:28.185 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:21:28.191 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
[INFO] [1671074492.887752677] [rclcpp]: Incoming request
j1vel_ref: 4.140940 j2vel_ref: -5.781005 j3vel_ref: 0.000000
[INFO] [1671074492.908745355] [rclcpp]: sending back response:
 [-1.848493, 2.121361, 0.000000]
```

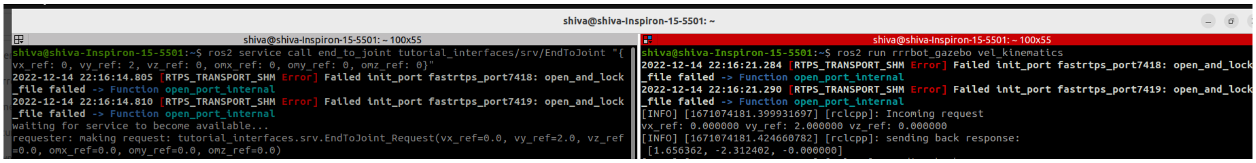
(Part 1) Joint Velocities to End effector velocity


shiva@shiva-Inspiron-15-5501: ~
shiva@shiva-Inspiron-15-5501: ~ 100x55
shiva@shiva-Inspiron-15-5501: ~ 100x55

```
shiva@shiva-Inspiron-15-5501:~$ ros2 service call end_to_joint tutorial_interfaces/srv/EndToJoint "{
vx_ref: 0, vy_ref: 2, vz_ref: 0, omx_ref: 0, omy_ref: 0, omz_ref: 0}"
2022-12-14 22:16:14.805 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:16:14.810 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
waiting for service to become available...
requester: making request: tutorial_interfaces.srv.EndToJoint_Request(vx_ref=0.0, vy_ref=2.0, vz_ref
=0.0, omx_ref=0.0, omy_ref=0.0, omz_ref=0.0)
```

```
shiva@shiva-Inspiron-15-5501:~$ ros2 run rrrbot_gazebo vel_kinematics
2022-12-14 22:16:21.284 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:16:21.290 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
[INFO] [1671074181.399931697] [rclcpp]: Incoming request
vx_ref: 0.000000 vy_ref: 2.000000 vz_ref: 0.000000
[INFO] [1671074181.424660782] [rclcpp]: sending back response:
 [1.656362, -2.312402, -0.000000]
```

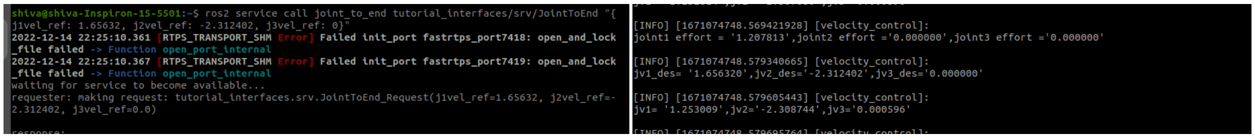
(Part 1)End effector velocities to joint velocities.


shiva@shiva-Inspiron-15-5501: ~
shiva@shiva-Inspiron-15-5501: ~ 100x55
shiva@shiva-Inspiron-15-5501: ~ 100x55

```
shiva@shiva-Inspiron-15-5501:~$ ros2 service call end_to_joint tutorial_interfaces/srv/EndToJoint "{
vx_ref: 0, vy_ref: 2, vz_ref: 0, omx_ref: 0, omy_ref: 0, omz_ref: 0}"
2022-12-14 22:16:14.805 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:16:14.810 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
waiting for service to become available...
requester: making request: tutorial_interfaces.srv.EndToJoint_Request(vx_ref=0.0, vy_ref=2.0, vz_ref
=0.0, omx_ref=0.0, omy_ref=0.0, omz_ref=0.0)
```

```
shiva@shiva-Inspiron-15-5501:~$ ros2 run rrrbot_gazebo vel_kinematics
2022-12-14 22:16:21.284 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:16:21.290 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
[INFO] [1671074181.399931697] [rclcpp]: Incoming request
vx_ref: 0.000000 vy_ref: 2.000000 vz_ref: 0.000000
[INFO] [1671074181.424660782] [rclcpp]: sending back response:
 [1.656362, -2.312402, -0.000000]
```

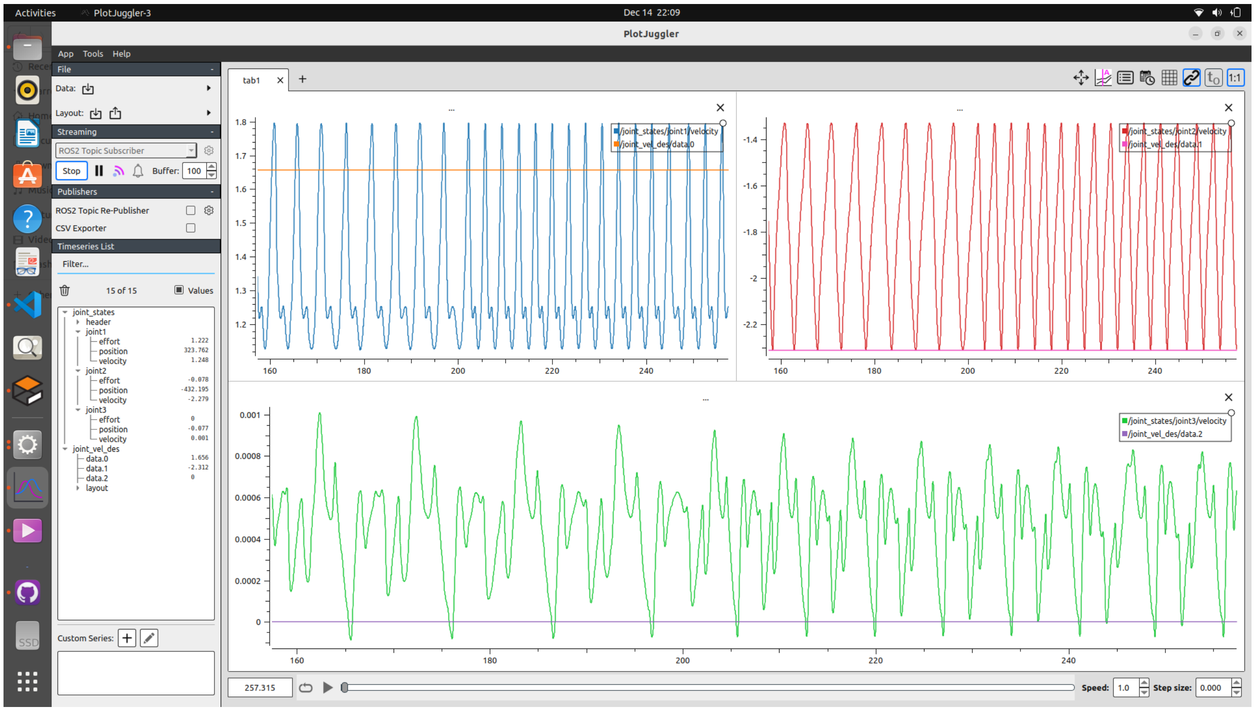
(Part 3) Joint velocity references for end effector of 2 in the Y direction



```
shiva@shiva-Inspiron-15-5501:~$ ros2 service call joint_to_end tutorial_interfaces/srv/JointToEnd "{
j1vel_ref: 1.65632, j2vel_ref: -2.312402, j3vel_ref: 0}"
2022-12-14 22:25:10.361 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7418: open_and_lock
_file failed -> Function open_port_internal
2022-12-14 22:25:10.367 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7419: open_and_lock
_file failed -> Function open_port_internal
waiting for service to become available...
requester: making request: tutorial_interfaces.srv.JointToEnd_Request(j1vel_ref=1.65632, j2vel_ref=-
2.312402, j3vel_ref=0.0)

response:
```

```
[INFO] [1671074748.569421928] [velocity_control]:
joint1 effort = '1.207813',joint2 effort ='0.000000',joint3 effort ='0.000000'

[INFO] [1671074748.579340665] [velocity_control]:
jv1_des= '1.656320',jv2_des='-2.312402',jv3_des='0.000000'

[INFO] [1671074748.579605443] [velocity_control]:
jv1= '1.253009',jv2='-2.308744',jv3='0.000596'

[INFO] [1671074748.579695764] [velocity_control]:
```

(Part 3) Reference Joint velocity and then resulting Joint velocity



```
joint_states
  header
  joint1
    effort      1.222
    position  323.762
    velocity    1.248
  joint2
    effort     -0.078
    position -432.195
    velocity   -2.279
  joint3
    effort      0
    position   -0.077
    velocity    0.001
joint_vel_des
  data.0        1.656
  data.1       -2.312
  data.2        0
  layout
```

(Part ￼3) Reference and actual joint velocity plots

<u>Result</u>: The robot end effector moves in the y-direction as per the video