Group Assignment – Part 2
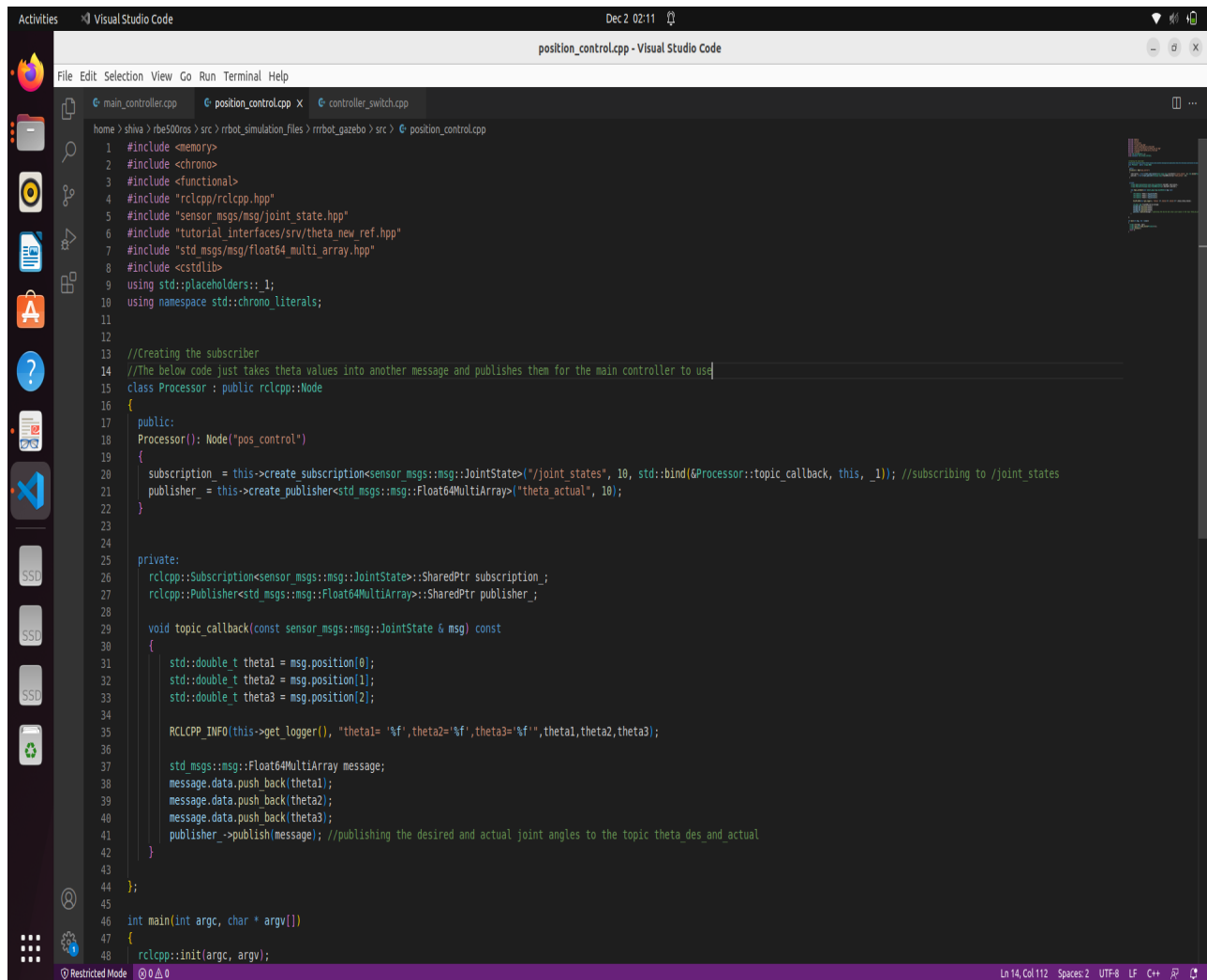**Shiva Surya Lolla**
Chaitanya Sriram Gaddipati
Ethan Wilke

Part 1.) The Code

The "Position control" file consists of a subscriber file that reads and organizes the recorded current positions of the joints and then sends them out in a message format through a publisher. Below is its code. (Please check code comments below for documentation)

"Main Controller" takes in the message from the "position_control" node and takes desired joint values through a service. It calculates the joint efforts through PD controllers for each joint. This calculated joint effort for each joint is then sent out through a publisher, eventually being read

from the Gazebo program for the simulation. Below is its code. (Please check code comments below for documentation)

```cpp
class NewProcessor : public rclcpp::Node
{
  public:
    NewProcessor()
    : Node("main_control")
    {
      //create a service to take desired joint values
      service_ = this->create_service<tutorial_interfaces::srv::PositionControl>("position_control", std::bind(&NewProcessor::jointref, 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 theta1_des;
    mutable double theta2_des;
    mutable double theta3_des;

    void jointref(const std::shared_ptr<tutorial_interfaces::srv::PositionControl::Request> request){
      //taking desired angle values from the client request from the terminal
      theta1_des = request->joint1_ref;
      theta2_des = request->joint2_ref;
      theta3_des = request->joint3_ref;
      //subscribing to actual joint angles
      subscription_1 = this->create_subscription<std_msgs::msg::Float64MultiArray>("theta_actual", 10,std::bind(&NewProcessor::topic1_callback, this, _1));
      publisher_1 = this->create_publisher<std_msgs::msg::Float64MultiArray>("/forward_effort_controller/commands", 10); //publishing joint efforts
      publisher_2 = this->create_publisher<std_msgs::msg::Float64MultiArray>("/theta_des", 10); //publishing desired joint values for plotting
    }
```

```cpp
    void topic1_callback(const std_msgs::msg::Float64MultiArray & msg) const
    {
      //extracting the actual joint values
      std::double_t theta1 = msg.data[0];
      std::double_t theta2 = msg.data[1];
      std::double_t theta3 = msg.data[2];
      RCLCPP_INFO(this->get_logger(), "theta1_des= '%f',theta2_des='%f',theta3_des='%f'",theta1_des,theta2_des,theta3_des);
      RCLCPP_INFO(this->get_logger(), "theta1= '%f',theta2='%f',theta3='%f'",theta1,theta2,theta3);

      //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 = 0.09; //joint 1 proportional gain
      double Kd1 = 0.09; //joint 1 derivative gain
      double Kp2 = 0.05; //joint 2 proportional gain
      double Kd2 = 0.06; //joint 2 derivative gain
      double Kp3 = 1200; //joint 3 proportional gain
      double Kd3 = 9; //joint 3 derivative gain

      //creating message for publisher_ to publish efforts
      std_msgs::msg::Float64MultiArray message;
      std_msgs::msg::Float64MultiArray message1;
```

```cpp
        //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(theta1_des-theta1)>epsilon)){
            e1 = theta1_des - theta1; //calculating error
            if(std::abs(theta1)<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(theta2_des-theta2)>epsilon)){
            e2 = theta2_des - theta2;
            if(std::abs(theta2)<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(theta3_des-theta3)>epsilon)){
            e3 = theta3_des - theta3;
            if(std::abs(theta3)<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(theta1_des);
        message1.data.push_back(theta2_des);
        message1.data.push_back(theta3_des);
        RCLCPP_INFO(this->get_logger(), "joint1e= '%f',joint2e='%f',joint3e='%f'",joint1_effort,joint2_effort,joint3_effort);
        //publishing joint efforts
        publisher_1->publish(message);
        //publishing desired angles taken from the service
        publisher_2->publish(message1);
    }
    rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr publisher_1;
    rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr publisher_2;
    rclcpp::Subscription<std_msgs::msg::Float64MultiArray>::SharedPtr subscription_1;
    rclcpp::Service<tutorial_interfaces::srv::PositionControl>::SharedPtr service_;
};

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

PART 2.) **The Results**
Initial condition of robot
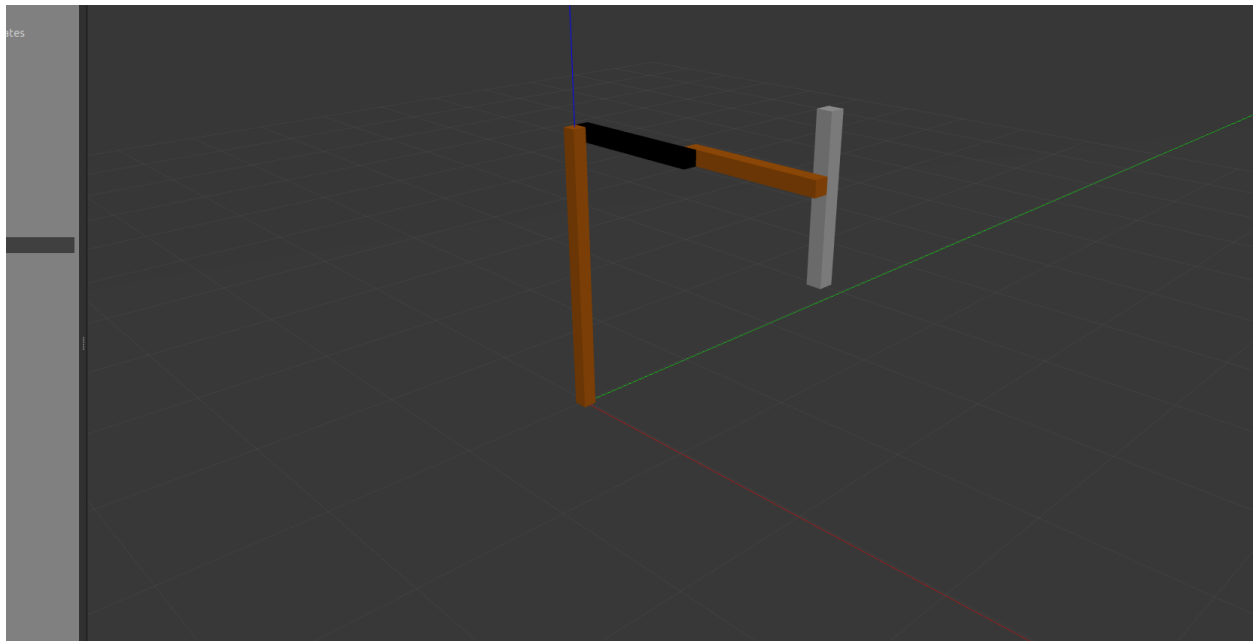
Joint Position Reference 1
Joint 1 reference value: 0.5
Joint 2 reference value: 1.57
Joint 3 reference value: -0.2
Screenshot attached below for service call

Graphs generated for a period of 60 seconds with sampling time 0.1 seconds. As we can see, the final values are close to the reference values
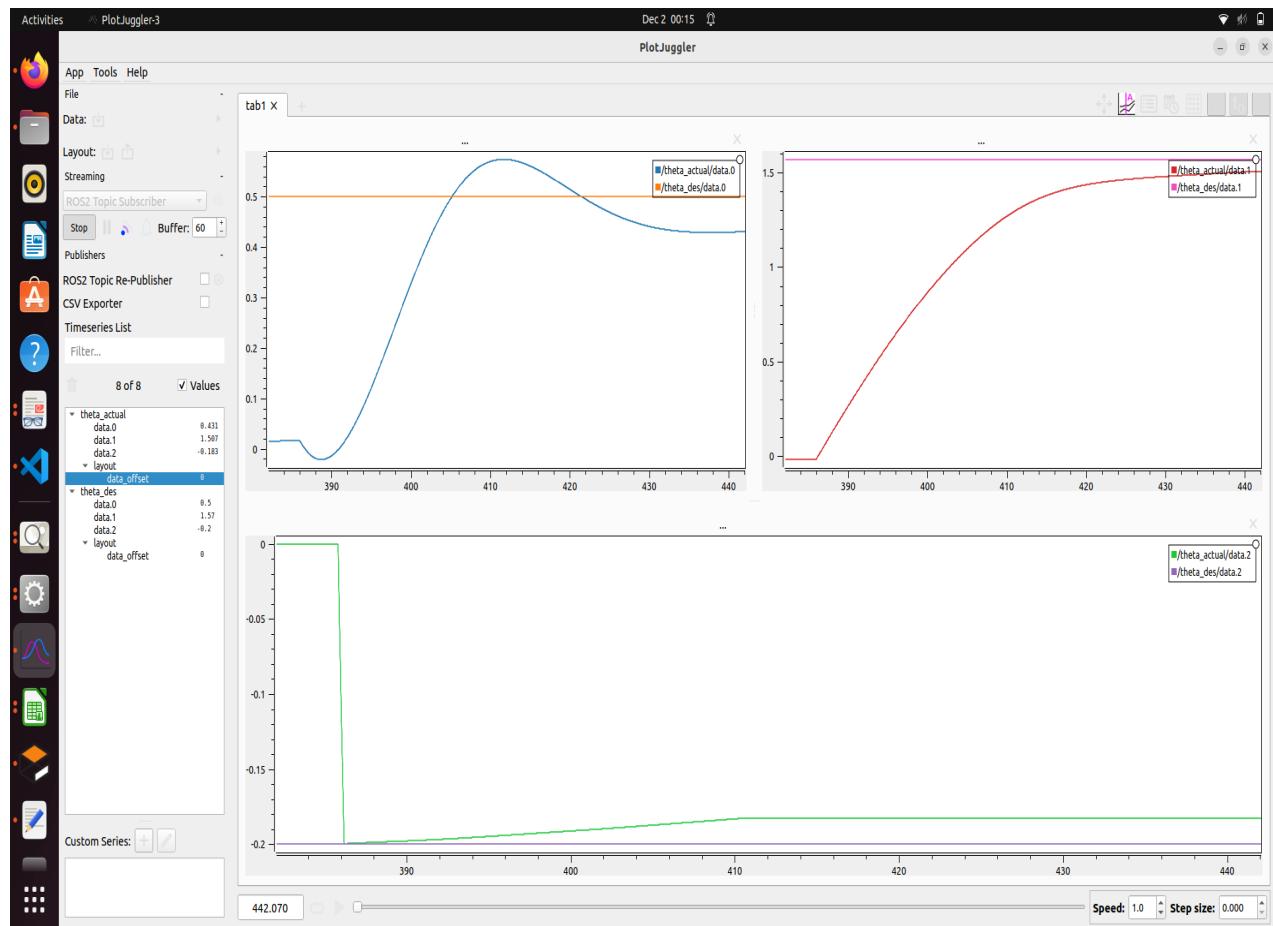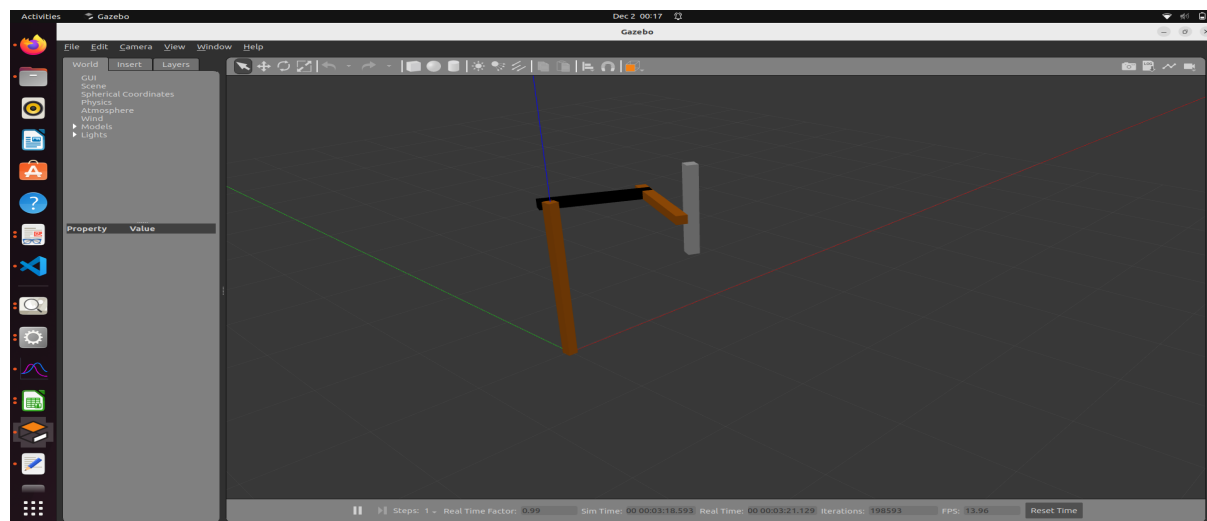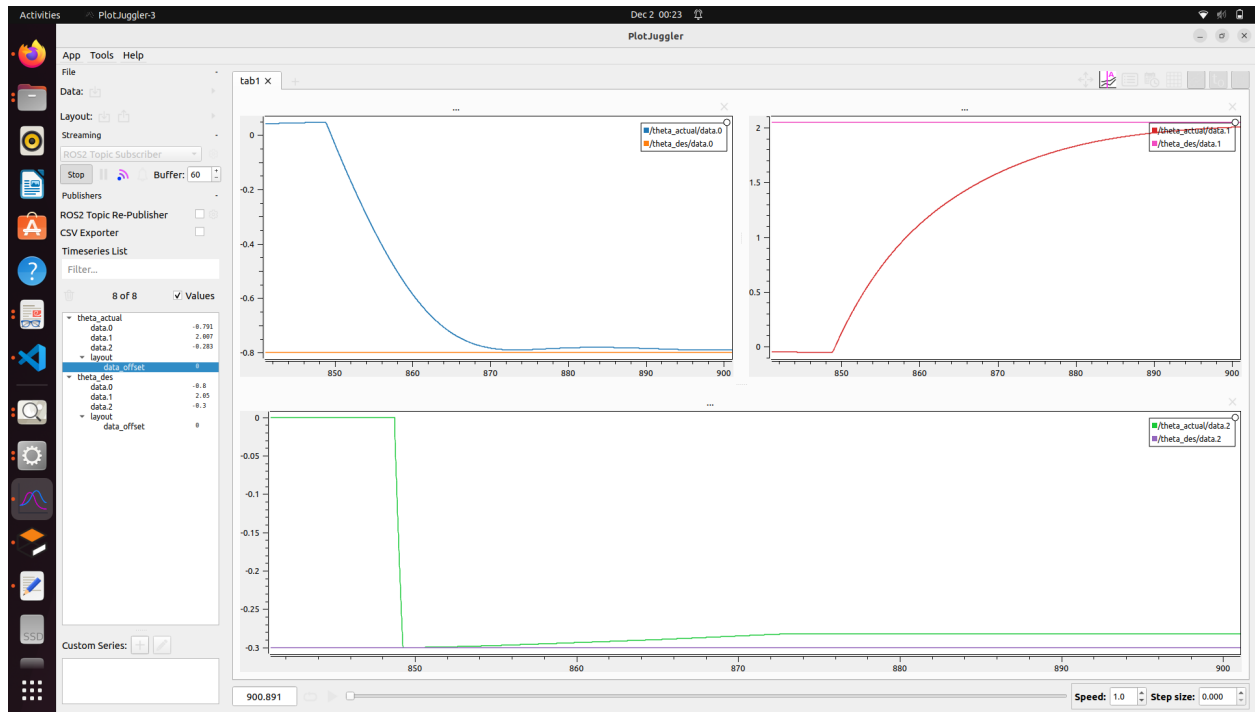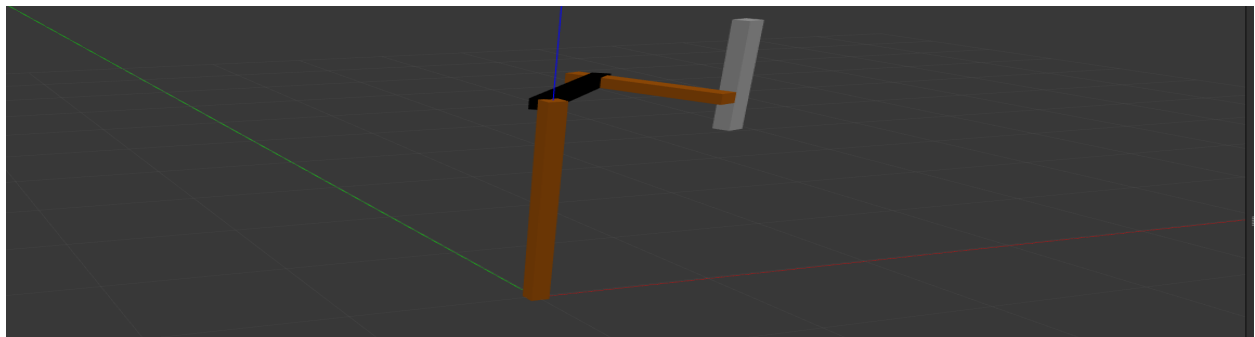

Figure: Joint 1 graph (top left), Joint 2 graph (top right), Joint 3 graph (bottom)

Final condition of the robot

Joint Position Reference 2
Joint 1 reference value: -0.8
Joint 2 reference value: 2.05
Joint 3 reference value: -0.3
Screenshot attached below for service call

Graphs generated for a period of 60 seconds with sampling time 0.1 seconds. As we can see, the final values are close to the reference values

Figure: Joint 1 graph (top left), Joint 2 graph (top right), Joint 3 graph (bottom)

Final condition of the robot

Joint Position Reference 3

Joint 1 reference value: 1.5

Joint 2 reference value: 1

Joint 3 reference value: -0.1

Screenshot attached below for service call

Graphs generated for a period of 60 seconds with sampling time 0.1 seconds. As we can see, the final values are close to the reference values
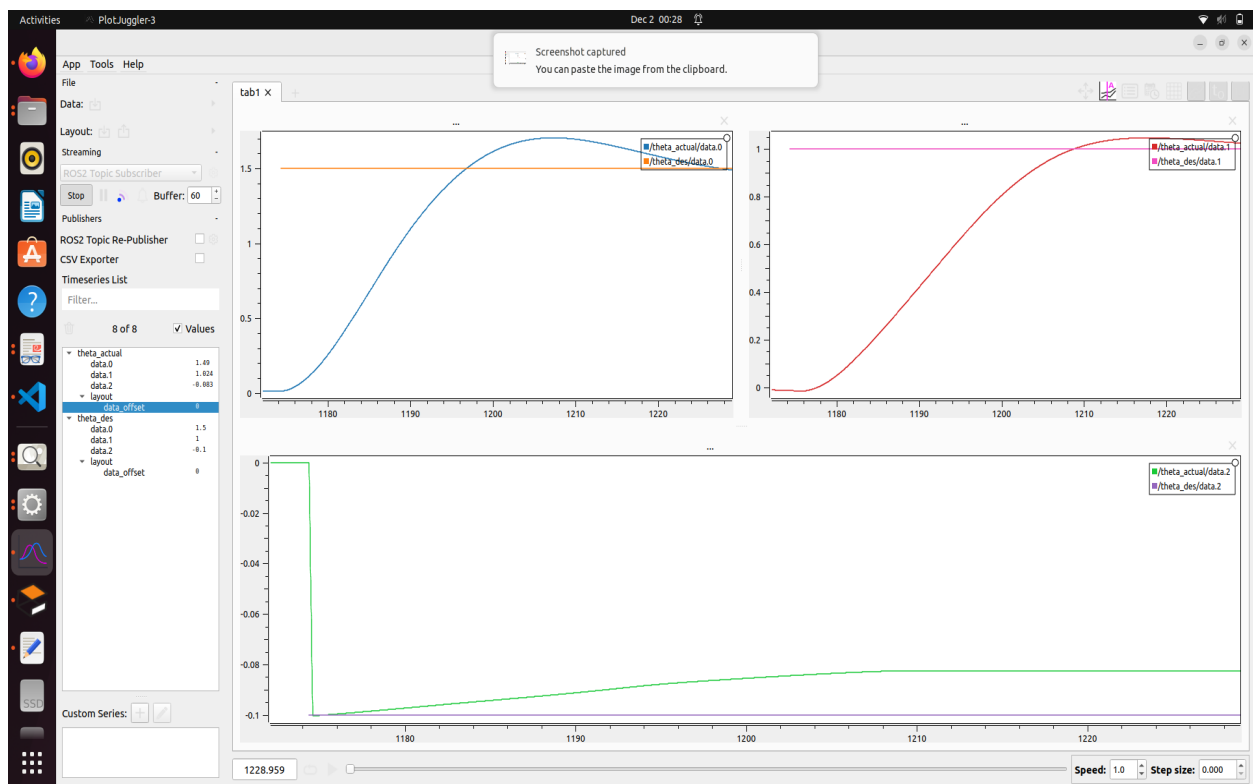

Figure: Joint 1 graph (top left), Joint 2 graph (top right), Joint 3 graph (bottom)

Final Position of the robot

From the above results it can be observed that there is some small steady state error in each joint value despite extensive tuning. This can be reduced further by using PID control instead.

PART 3.) **Instructions:**

These are the instructions to run the code shown above. Create a new workspace and keep the two packages in the submission file into a src folder. Build the workspace. To use the position control node follow the below commands.
I. Open a terminal and launch the gazebo simulation using:
ros2 launch rrrbot_gazebo rrbot_world.launch.py
Note: The package name is 'rrrbot_gazebo' and not 'rrbot_gazebo'
II. Open another terminal pane and now switch the controller to effort controller using:
ros2 run rrrbot_gazebo switch
III. Open another terminal pane for service call using:

ros2 service call position_control tutorial_interfaces/srv/PositionControl "{joint1_ref: 0.5, joint2_ref: 1.57, joint3_ref: -0.2}"
IV. In another terminal pane run the controller using
ros2 run rrrbot_gazebo pos_control
V. In another terminal pane run the main controller using
ros2 run rrrbot_gazebo main_control

Note: While giving the sample reference joint values make sure the joint3_ref values are negative. This is due to the design of the robot prismatic joint in the urdf file.