TEK4030
Mandatory project - Motion Control

- Group 2 -

Eivind Brastad Dammen, Øyvind Soma, Song Luu, Stephen Hangluah and Sjamil Gazimagamaev

Introduction

In this assignment we implemented a joint space PD controller with gravity compensation on the CrustCrawler simulation.

The block scheme for the PD controller can be seen in the figure below *(Figure 1)*.

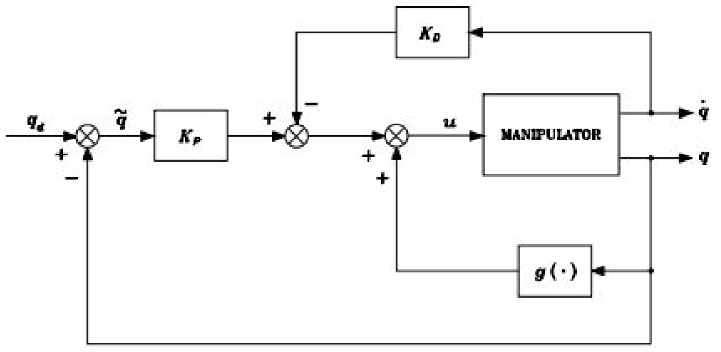


Figure 1: Block scheme of joint space PD control with gravity compensation

In order to implement the controller was it first necessary to set up the CrustCrawler simulation environment. This was achieved by following the given guidelines in the assignment description.

After setting up the simulation environment, the controller was implemented using the control law given in *Equation 1* (found on page 329 in the curriculum book).

$$u = g(q) + K_P\tilde{q} - K_D\dot{q}, \qquad (1)$$

In order to compute the control law the following parameters had to be found:

- **q:** The parameter is given as the current position of each of the joints.
- **g:** The parameter represents the gravity vector.
- **Kp and Kd:** These parameters are the control gains given as diagonal matrices.

- **q̃:** This parameter is the difference between ***qd*** and the actual position ***q*** as shown in *Equation 2*, where ***qd*** is given as the desired position of the joints.

$$\tilde{q} = qd - q \qquad (2)$$

- **q̇:** The parameter is given as the velocity of the of the joints.

These parameters were then used in order to compute the controller given in *Equation 1*.

In the next subsection it will be given a more detailed description on how the controller was implemented and which resources were used in order to achieve the desired results.

<u>Method</u>

We created a package called ***pd_controller***, which contains; ***pd_controller.cpp***, ***pd_controller.launch*** and a CMakeList.

1. **pd_controller.cpp:**

   This file contains the implementation and usage of the controller given in *Equation 1*, which was achieved in the following way:

   Firstly, we created a function called *getJointStates*, which was used to find the necessary parameters, compute the controller and publish the results.

   The parameter ***q*** was found by giving it the position values of the joint states, while the parameter $\dot{\boldsymbol{q}}$ was given the velocity values.

   In order to find the gravity parameter ***g***, was the function *getGravityVector* (function which was implemented and given to us) used with the parameter ***q*** as input.

   We then assigned initial values for the following parameters; ***Kp***, ***Kd*** and ***qd*** (for each of the joints).

   These parameters were then used in *Equation 1* to compute ***u***.

   When tuning gain-parameters, we started with only proportional gain, slowly increasing until the system had a steady oscillation. Then we increased the derivative gain until we had somewhat acceptable values.

   Finally, after computing the control effort ***u*** in our controller, the output was sent back to the simulation as std_msgs. The whole system can be observed in *Figure 2*.

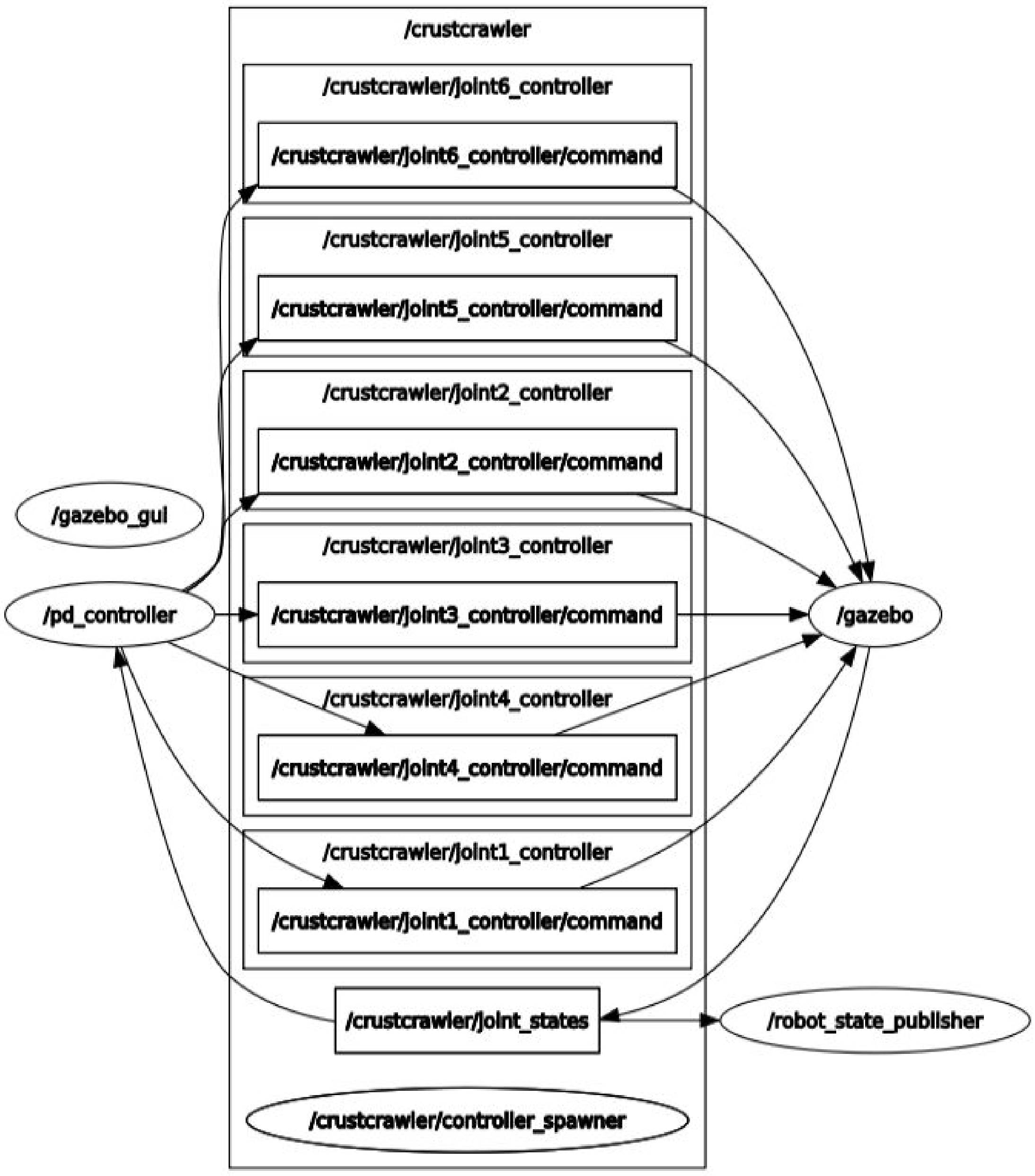

/crustcrawler
/crustcrawler/joint6_controller
/crustcrawler/joint6_controller/command
/crustcrawler/joint5_controller
/crustcrawler/joint5_controller/command
/crustcrawler/joint2_controller
/crustcrawler/joint2_controller/command
/gazebo_gui
/crustcrawler/joint3_controller
/pd_controller
/crustcrawler/joint3_controller/command
/gazebo
/crustcrawler/joint4_controller
/crustcrawler/joint4_controller/command
/crustcrawler/joint1_controller
/crustcrawler/joint1_controller/command
/crustcrawler/joint_states
/robot_state_publisher
/crustcrawler/controller_spawner

Figure 2: Block scheme of the nodes in ROS

2. **pd_controller.launch:**

   A launch file ***pd_controller.launch*** was created in order to start the controller node ***pd_controller*** from a separate terminal.

3. **CMakeList:**

   The CMakeList contains the necessary libraries, dependencies and packages needed in order to execute the ***pd_controller*** node.

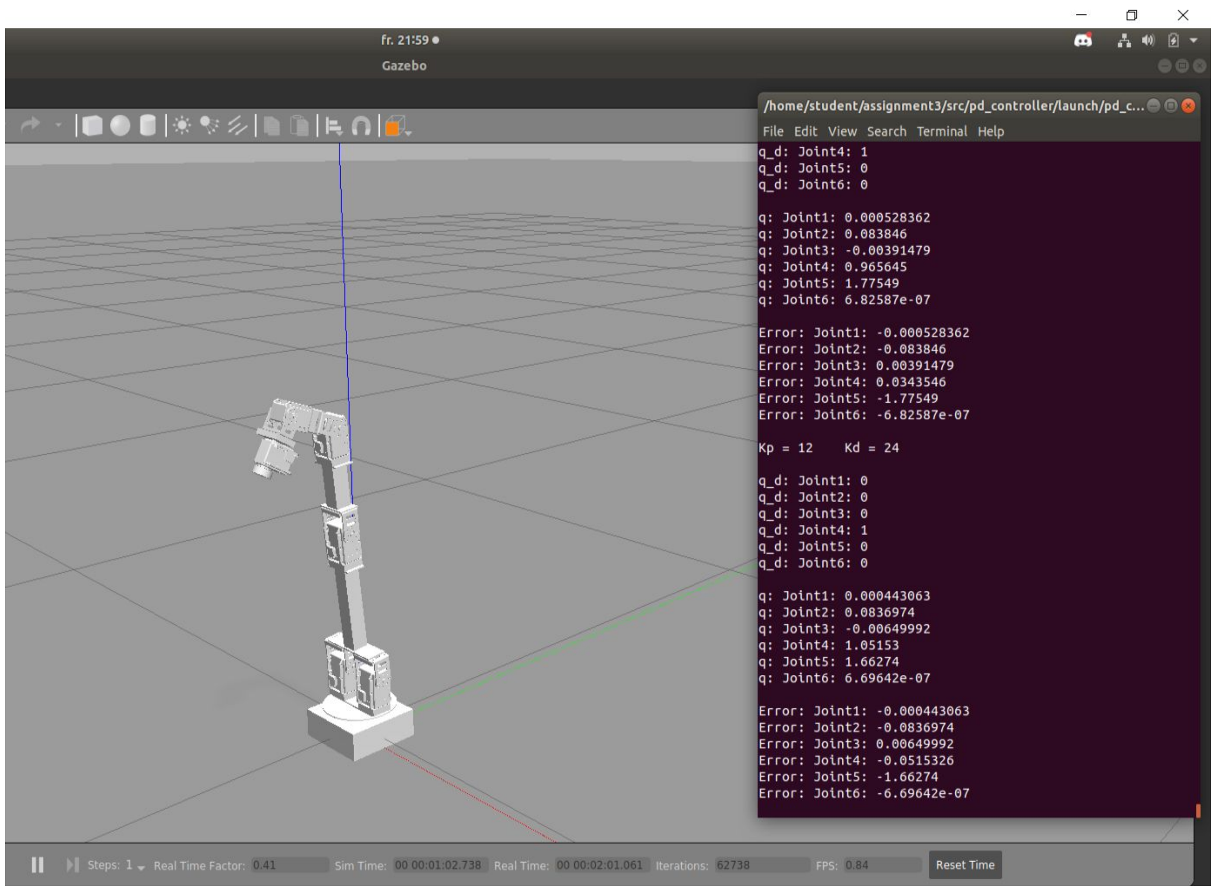
Results

In order to test the performance, we decided to set the desired joint positions to be:

qd = [0,0,0,1,0,0]

The controller was then tested with various ***Kp*** and ***Kd*** values, which gave the following results:

After a bit of testing, we were able to find parameters which resulted in errors with low values, this was the case for all the joints except for one (which can be seen in the screenshot provided below).


fr. 21:59
Gazebo
/home/student/assignment3/src/pd_controller/launch/pd_c...
File Edit View Search Terminal Help
Steps: 1
Real Time Factor: 0.41
Sim Time: 00 00:01:02.738
Real Time: 00 00:02:01.061
Iterations: 62738
FPS: 0.84
Reset Time

```
q_d: Joint4: 1
q_d: Joint5: 0
q_d: Joint6: 0

q: Joint1: 0.000528362
q: Joint2: 0.083846
q: Joint3: -0.00391479
q: Joint4: 0.965645
q: Joint5: 1.77549
q: Joint6: 6.82587e-07

Error: Joint1: -0.000528362
Error: Joint2: -0.083846
Error: Joint3: 0.00391479
Error: Joint4: 0.0343546
Error: Joint5: -1.77549
Error: Joint6: -6.82587e-07

Kp = 12    Kd = 24

q_d: Joint1: 0
q_d: Joint2: 0
q_d: Joint3: 0
q_d: Joint4: 1
q_d: Joint5: 0
q_d: Joint6: 0

q: Joint1: 0.000443063
q: Joint2: 0.0836974
q: Joint3: -0.00649992
q: Joint4: 1.05153
q: Joint5: 1.66274
q: Joint6: 6.69642e-07

Error: Joint1: -0.000443063
Error: Joint2: -0.0836974
Error: Joint3: 0.00649992
Error: Joint4: -0.0515326
Error: Joint5: -1.66274
Error: Joint6: -6.69642e-07
```

Figure 3: Final result

We're not sure why joint5 has such a high error, but we believe it could be partly fixed with a filter. Since there seemed to be some noise on this specific joint. This specific joint oscillated quite a bit even with a decent amount of tuning.

When tuning the gains, we first started with a low proportional-gain, slowly increasing it. Until we had a stable oscillation. Then we raised the derivative gain til it got pretty steady and returned a low error.