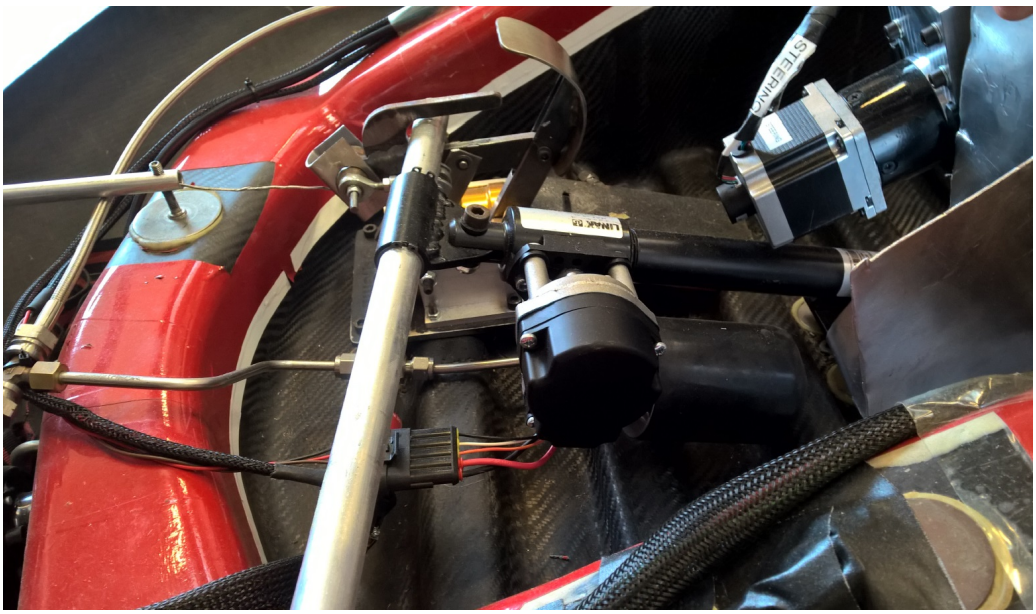

Further development and optimization of an autonomous braking system for the Ecocar

Special Course - Elective - 5 ECTS

Student:
Raffaele Barucci (s161825)

Supervisors:
Jesper Schramm
Claus Suldrup Nielsen

Autumn & Winter 2018

Contents

Chapter 1: Introduction

One of the weak points of the Ecocar 2018 was represented by the autonomous braking system. The original solution consisting in a closed-loop pressure control system didn't guarantee satisfactory performance. The root cause was represented by the position controller of the linear actuator that was used to apply pressure on the break pedal, which was designed for a different type of task. For this reason, before the competition a workaround was developed. The temporary solution was operated in an open-loop control scheme with a look-up table with different pressure levels relative to the position of the actuator. The table required frequent calibration. Although the performance was better than the previous solution, it was of primary importance to improve the performance of the autonomous braking system for the Ecocar 2019 competition.

Chapter 2: Understanding the previous system

2.1 Braking system

The mechanical system is composed by a linear actuator from Linak (model 303100-4010020N). The actuator exerts pressure on a braking pedal. A linear guiding mechanism allows the pilot to brake at any moment and override the actuator. The pedal pushes two hydraulic cylinders. One of the cylinders applies pressure on the oil that transmits the pressure to the brake calipers in the front whereas the other cylinder is responsible for the back brakes.

On the electrical side, a position controller from Linak (model TR-EM-288-SAF) uses the position feedback from a potentiometer built in the linear actuator. Two pressure sensors from Danfoss (model AKS32-060G2003) record the pressure in the front and back brakes. The pressure signals are sampled by a Teensy micro-controller. Figure 2.1 shows the braking system schematically.

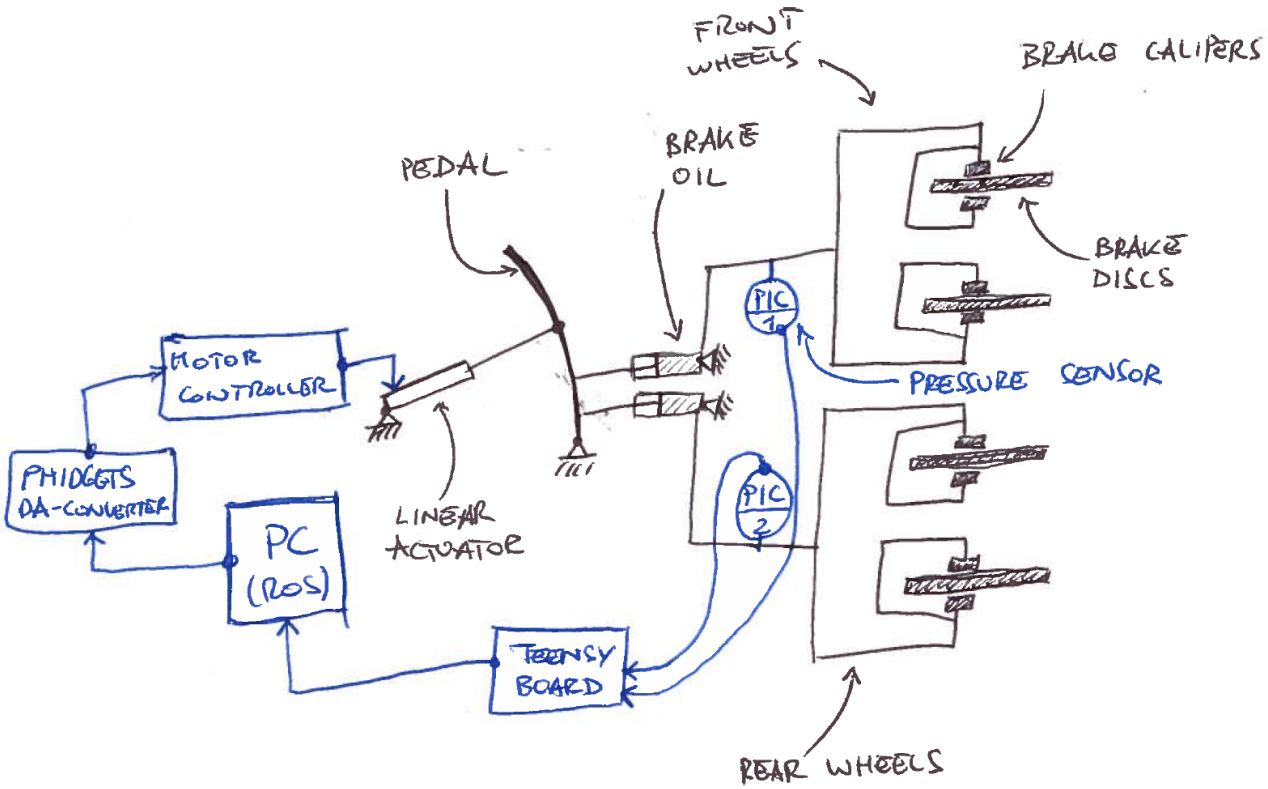

Figure 2.1: Braking system: in black the mechanical system and in blue the electrical

2.2 Control architecture

A cascade control architecture is chosen to control the pressure. A PID pressure controller, which runs on the main computer in a real-time node in ROS, drives the PID position controller. The position controller takes an analog voltage which is generated by a D/A converter connected to the computer through USB (Phidgets 1002). Figure 2.2 shows the control scheme.

Cascade control is an effective solution when the dynamics of the inner loop/slave loop, which is the linear actuator, are faster than the dynamics of the outer loop/master loop, which is the hydraulic braking system. Furthermore, the inner loop disturbances are less severe than the outer loop disturbances. Therefore a cascade architecture should provide a better performance than a single loop controller that is based only on pressure feedback and which acts directly on the voltage of the DC motor of the linear actuator. In addition, running the linear actuator in open-loop wouldn't allow to set limits to the position of the actuator.

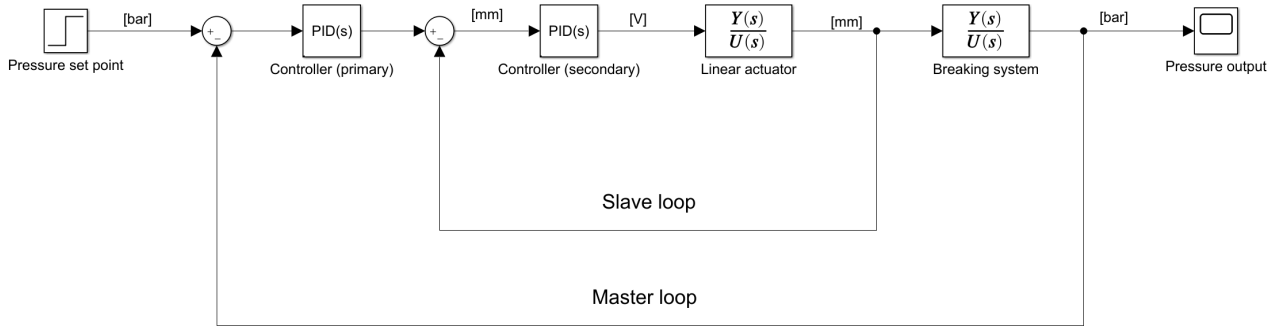

Figure 2.2: Cascade control scheme of the braking system

2.3 Position control of the linear actuator

As mentioned previously, a Linak position controller with potentiometer feedback (TR-EM-288-SAF) was used for the past competition. The first step was to remove the actuator and the controller from the car, analyze its performance and identify the weak points. An Arduino micro-controller with LIFA (LabVIEW Interface for Arduino) was used to acquire the position and set-point signals (see figure 2.3).

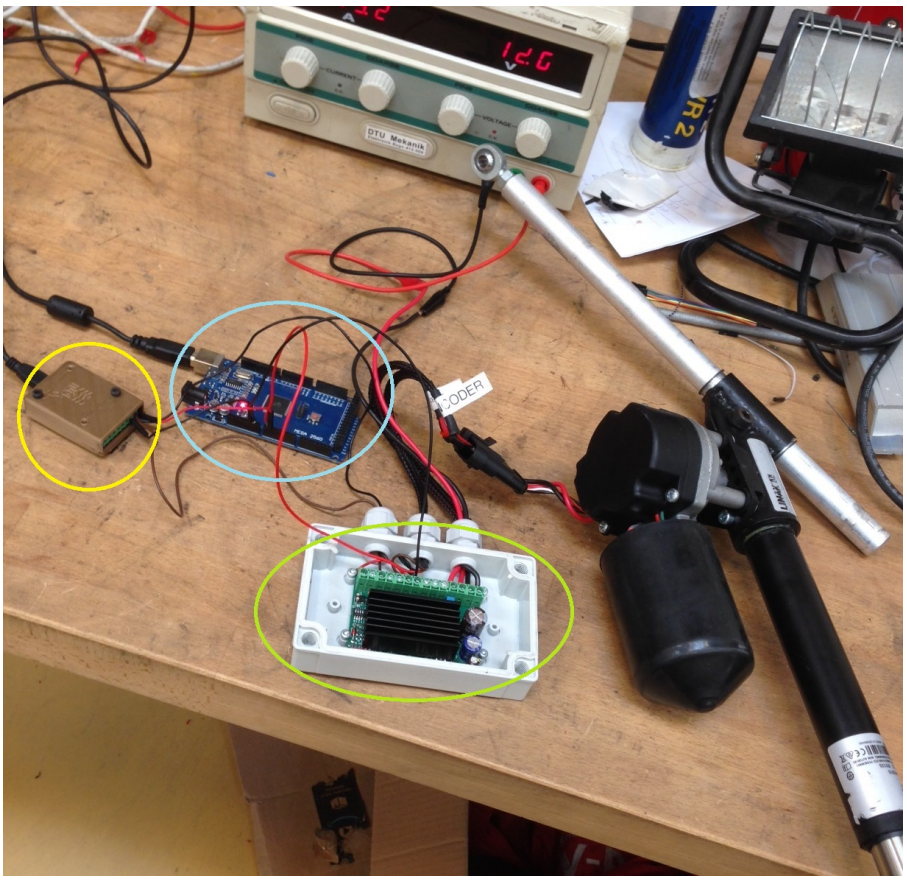

Figure 2.3: Test setup for the Linak controller (green circle), consisting of an Arduino board (blue circle) and a Phidgets digital-analog converter to generate a voltage signal for the position

A LabVIEW program was created to analyze the step response of the system. It became quickly clear, that the Linak controller was not driving the actuator to the given reference point (see picture 2.4). Furthermore, the controller was ignoring small changes in the reference signal (approximately 0.05V). An additional problem is that the controller doesn't accept a new reference input until the previous reference point is reached. All this issues make the Linak position controller unsuitable to be used for pressure control.

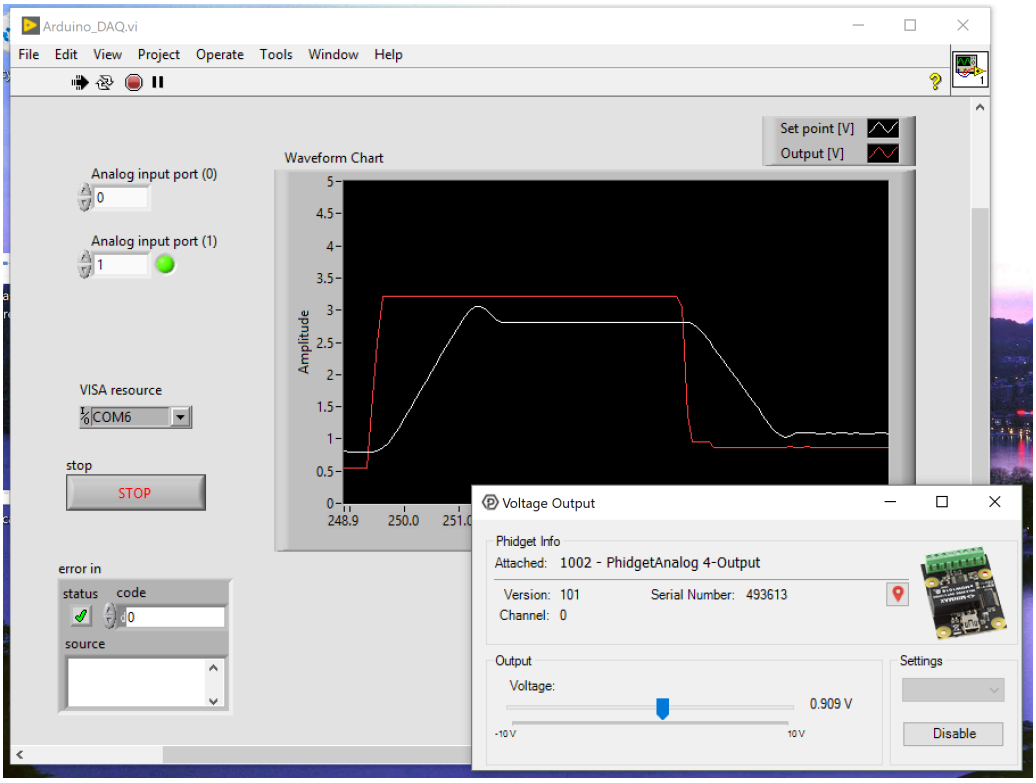

Figure 2.4: Step response of the linear actuator when using the Linak TR-EM-288-SAF controller

One last attempt was done with a different motion controller from Linak (model TR-EM-288-H) without feedback, which was bought as a spare part for the past competition. The controller doesn't work with an analog input voltage for positioning, however, it accepts a digital signal for moving forward/backward or braking and analog signal for the speed. A LabVIEW VI was used to drive the controller forward or backward until the desired reference position was reached. The analog input port of an Arduino was configured to get the feedback signal from the linear actuator's potentiometer. The performance of this system was acceptable at low speed and the tracking of the reference point seemed slightly improved compared to the previous controller (see picture 2.5).

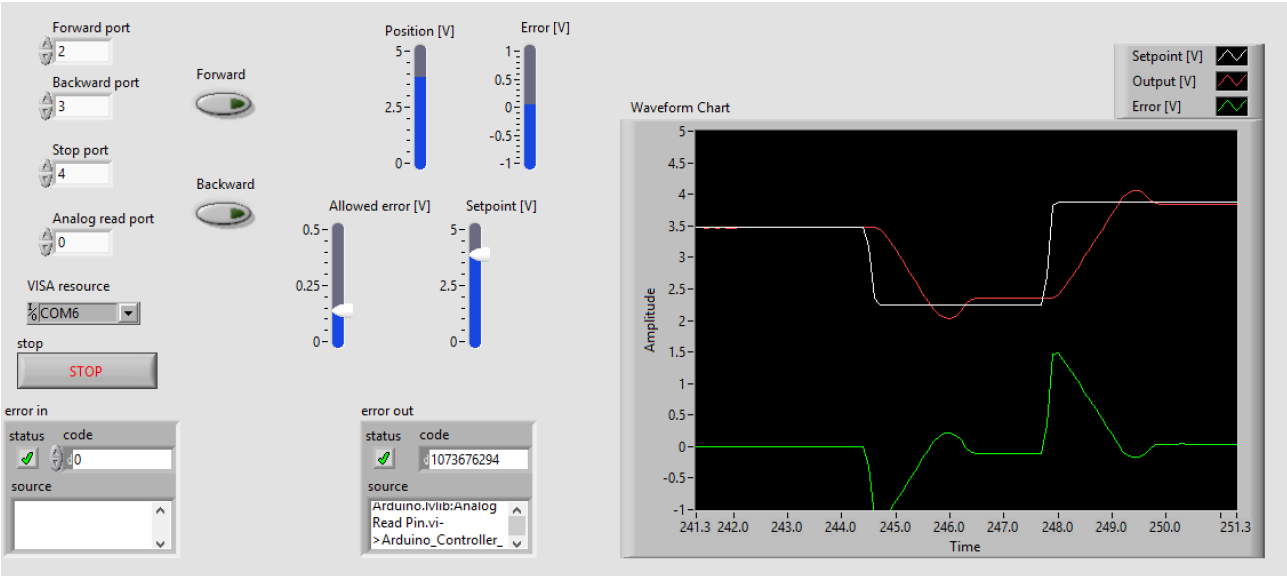

Figure 2.5: Step response with the Linak TR-EM-288-H controller

However at maximum speed the actuator became only marginally stable and kept oscillating around the reference point (see picture 2.6). This is because the actuator was either moving forward or backward at maximum voltage. The system could have been improved by including the speed regulation in the LabVIEW controller. However, it was decided that buying a new, highly configurable, motor controller with feedback would have been a more robust and less time consuming solution.

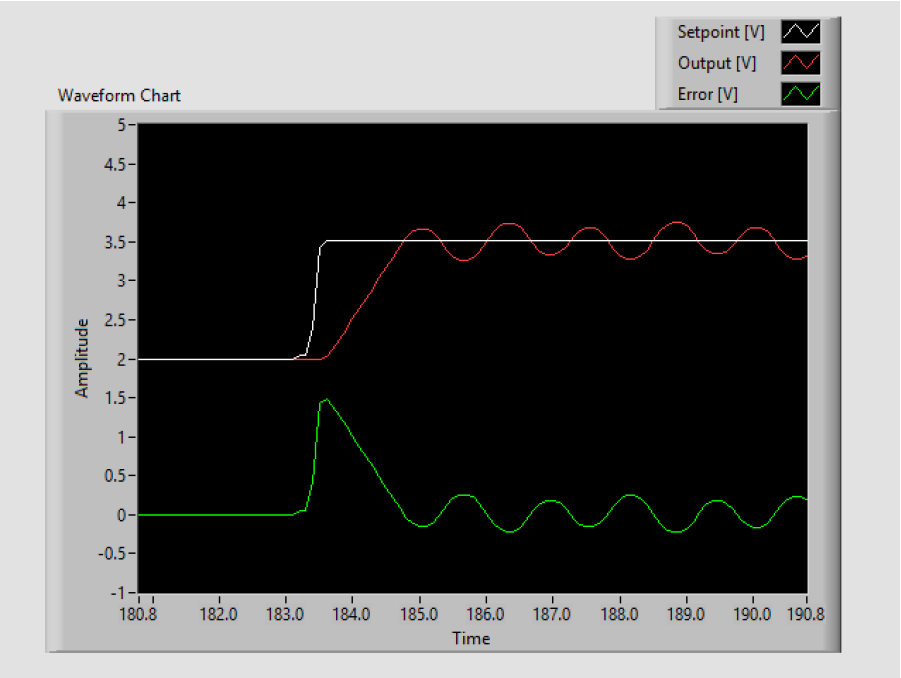

Figure 2.6: Step response with the Linak TR-EM-288-H controller

2.4 Pressure sensor

The pressure sensors from Danfoss (model AKS32 - 060G2003) work in a range from -1 to 34 bars and return a voltage signal between 1 and 5 Volts. The equation to obtain the pressure is therefore the following:

$$pressure = -\frac{39}{4} + \frac{35}{4} \cdot voltage \tag{2.1}$$

The pressure controller that was designed for the past competition used the pressure level of the rear brakes only. To improve the robustness of the system it would be desirable to use the maximum level of pressure of the front or rear brakes. The mechanical team is responsible for ensuring an accurate calibration of the front and rear hydraulic circuit.

Chapter 3: Choice of a new motion controller and integration in the system

3.1 Choice of a new controller

As mentioned in section 2.3, the performance of the Linak motion controller was insufficient. It was therefore necessary to find a new controller which fulfilled the requirements. The specifications of the linear a actuator and of the old controller were analyzed (see pictures 3.1 and 3.2). From the specifications it can be seen that the old controller is able to handle a current peak at start of the DC motor of 30A. The nominal voltage of the actuator is 12V and its current at full load reaches 20A.

Features

- *Power supply:* *10 - 35 V DC (ripple-max. 30%)*
- *Max. power limit:* *15 A duty cycle 100 %*
  30 A at start
- *Power limit, setting:* *0.1 - 20 A*
- *Ramp times (start/stop):* *0 - 5 seconds*
- *PWM frequency:* *2 kHz*
- *Controller input (Speed-2):* *0 - 5 V = 0 - 100 % speed*
- *Controller inputs:* *4 - 30 V = ON*
  0 - 1 V = OFF
- *Input impedance:* *10 k ohm*
- *Operating temp.: (Ta):* *- 40 - 60°C*

Figure 3.1: Specifications of the Linak TR-EM-288-H controller

Technical specifications:

New type	Spindle pitch (mm)	Thrust max. Push (N)	Thrust max. Pull (N)	*Self-lock max. With/without brake (N)	Typical speed 0/full load (mm/s)	Stroke length (mm)								Typical amp. at full load 12V	24V
307xx0-4xxxx0/5xx	12	1000	1000	1000/0	48/24	50	100	150	200P	250	300	350	400P	14	7
303xx0-4xxxx0/5xx	9	1500	1500	1500/400	42/20	50	100	150P	200	250	300P	350	400	14	7
302xx0-4xxxx0/5xx	6	2000	2000	2000/500	18.5/14	50	100P	150	200P	250	300	350	400P	14	7
301xx0-xxxxx0/5xx	3	3000	3000	3000/3000	16/9	50P	100P	150	200P	250	300	350	400	14	6.4
307xx0-4xxxx1/2xx	12	1000	1000	1000/0	65/35	50	100	150	200P	250	300	350	400P	20	10
303xx0-4xxxx1/2xx	9	1800	1800	1800/0	52/25	50	100	150P	200	250	300P	350	400	20	10

Figure 3.2: Specifications of the Linak 303100-4010020N actuator

A highly configurable DC motor controller from Pololu was chosen for the task. The Pololu Jrk G2 18v27 USB Motor Controller with Feedback operates in a range of 6.5 to 30V and can deliver continuous output currents up to 27 A without a heat sink (picture 3.3). Furthermore, it offers limitation of the current output through current chopping, which is an interesting feature due to the fact that the linear actuator is overdimensioned and can potentially damage the

structure of the car if extended too far. A significant amount of parameters can be changed through the software and by connecting the controller to a computer via USB [1].

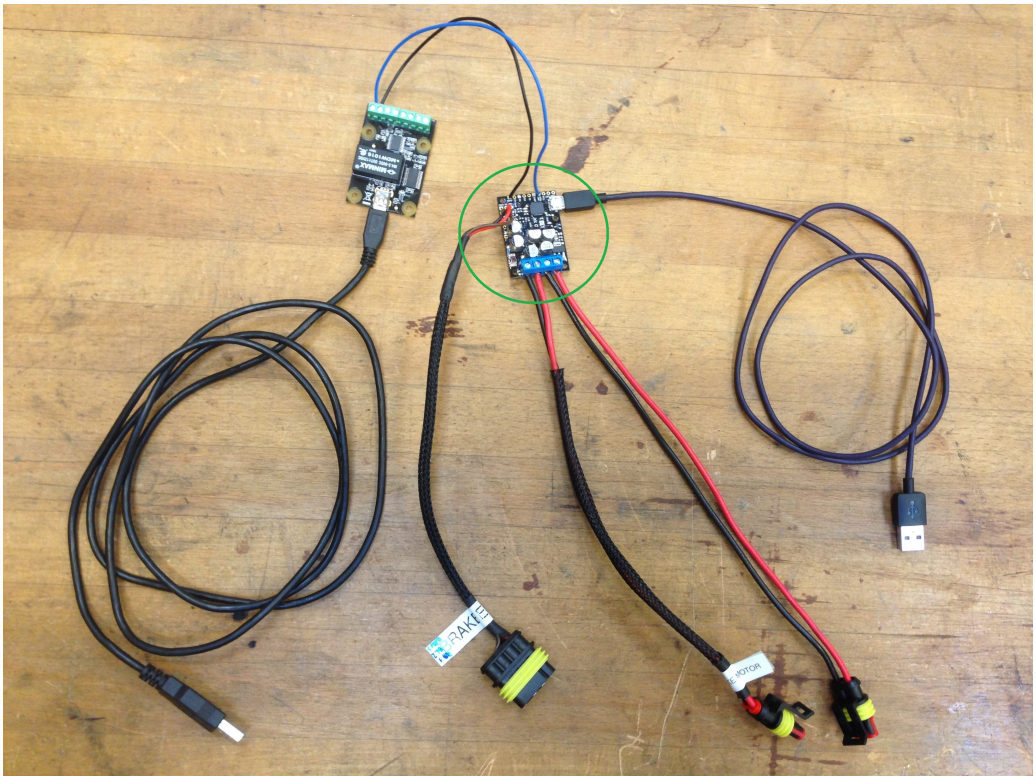

Figure 3.3: Pololu Jrk G2 18v27 USB Motor Controller with Feedback (green circle)

3.2 Configuration of the new controller

The Pololu controller was reatively easy to setup up with a configuration software. The first step was to define the limits of the linear actuator with the feedback setup wizard (see figure 3.4).

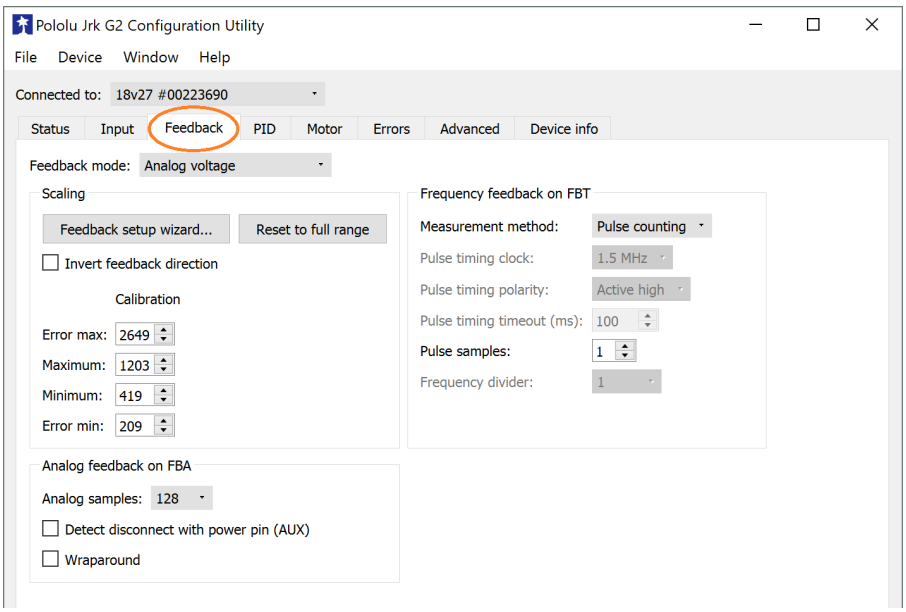

Figure 3.4: Configuration of the PID parameters of the new motion controller

The PID parameters were defined empirically by looking at the step response (see figures 3.5 and 3.6).

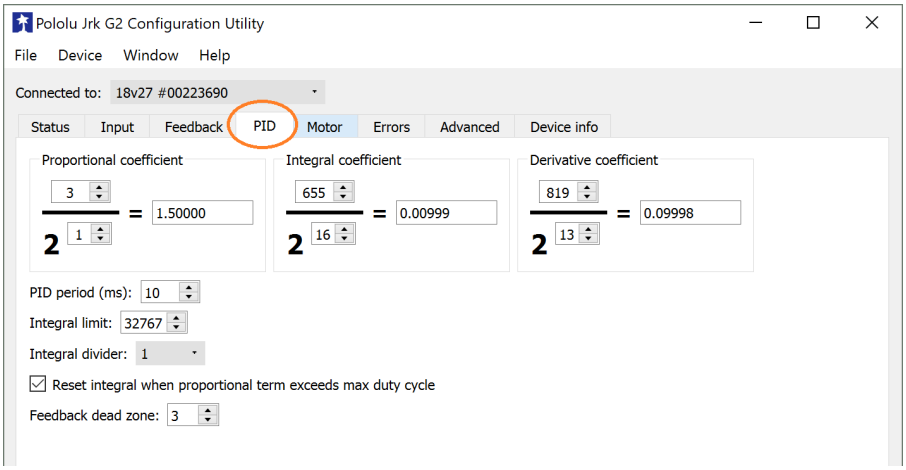

Figure 3.5: Configuration of the PID parameters of the new motion controller

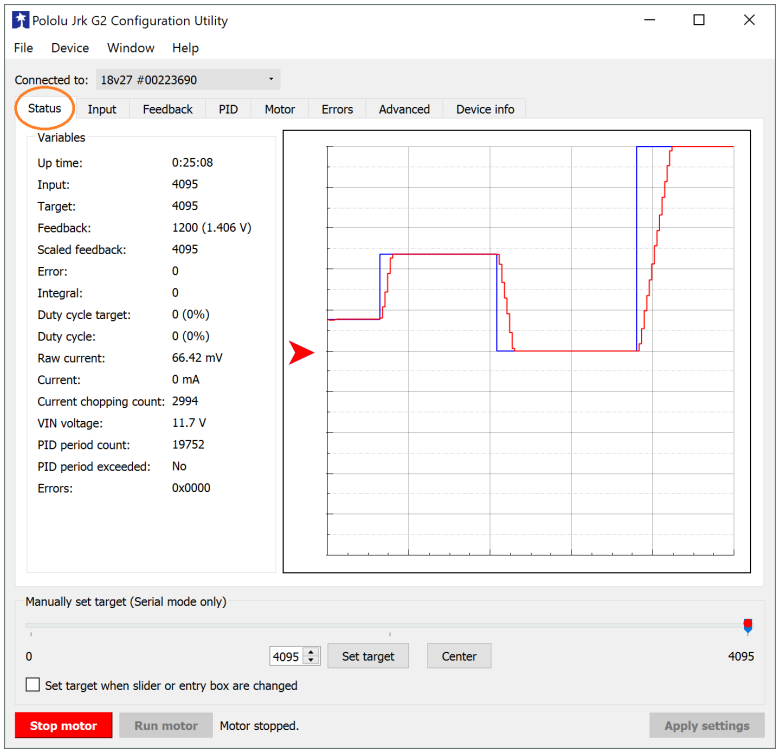

Figure 3.6: Step response of the new motion controller

The last step was to limit the motor current to avoid damage to the brake structure of the car (see figure 3.7). The current limiting feature is not very accurate (variations of up to 20% are possible according to Pololu), however, it is sufficient for limiting the actuator's force before damaging the car.

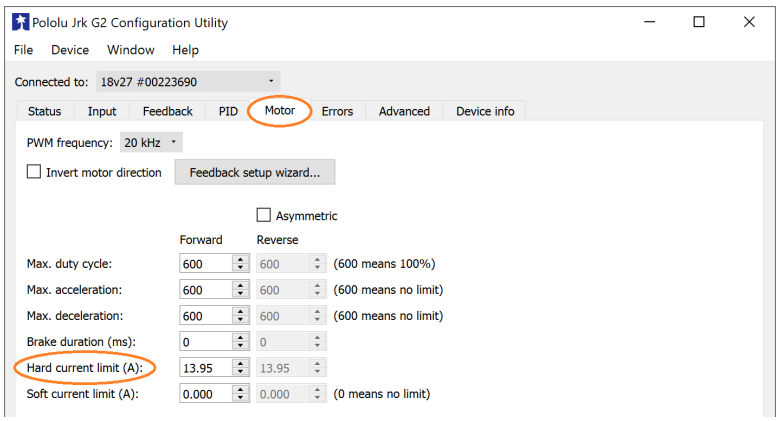

Figure 3.7: Step response of the new motion controller

The values of the parameters in the screen shots might differ from the ones currently saved on the controller. IMPORTANT DETAIL: THE PARAMETERS OF THE CONFIGURATION PROGRAM CAN BE EXPORTED IN A TXT FILE AND IMPORTED WITH FILE–> SAVE/OPEN SETTINGS.. I MADE A COPY OF THE SETTINGS IN catkin_ws/src/brake

3.3 Closing the pressure control loop

A ROS node containing a pressure controller was already programmed from a previous project. However, nobody in the team that had worked on ROS was left (a meeting with a team member who worked on the system was arranged at a later point). Therefore to close the loop of the pressure control system it was decided to create a temporary LabVIEW controller, use an Arduino microcontroller to acquire the data from the pressure sensor and the Phidgets device to generate a position signal for the motor controller. This solution was used for the prototyping of a controller. It became quickly clear that one of the main problems for controlling the system was the dead-zone caused by the guiding mechanism that allows the pilot to override the actuator (see picture 3.8.

Figure 3.8: Guiding mechanism that allows the pilot to override the autonomous brake system

Before being able to apply any force on the brakes the linear actuator needs to extend a couple of centimeters. Once the actuator has a stable contact with the pedal and the force necessary to move the brake calipers towards the discs is reached, the pressure starts growing exponentially (see picture 3.9). Furthermore, the pressure is initially negative (about -0.5bar) because of the springs of the brake calipers that act in the opposite direction causing vacuum.

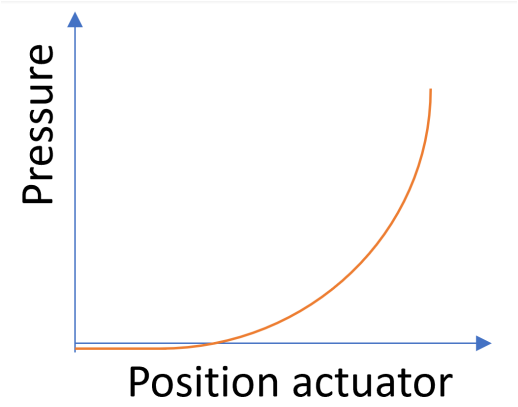

Figure 3.9: Characteristic curve of the pressure

The integral action of the PID controller can overcome the dead-zone at the beginning of the curve. However, in order to quickly cover the distance, high controller gains are required which would result in a very aggressive or even unstable response once a certain level of pressure is reached (right half of the curve in figure 3.9). In order to solve the problem it was decided to minimize the backlash between the actuator and the pedal, however, some tolerance is required to make sure that when the actuator is retracted no pressure is applied on the braking system. Furthermore, gain scheduling of the PID controller was implemented. Gain scheduling is a solution for controlling non-linear systems which involves different PID parameters for the different operating points of the system. The pressure level is used to determine what operating region the system is currently in. An aggressive set of gains is used from the region where there is no pressure at all until there is a level of 0.05 bar. At that point the PID gains are reduced to a more appropriate level.

The system was tested and the PID parameters were tuned empirically on the system. Figure 3.10 shows different step responses starting from no pressure to 5, 10, 15 and 20bar. There's a significant overshoot due to the dead-zone explained previously which causes the integrator to accumulate a high value which needs be reduced once the pressure starts rising exponentially. Furthermore, at pressure levels higher than 15bar some disturbances can be seen once the system settles. The mount where the brake pedal is fixed on the carbon chassis doesn't seem rigid enough when the linear actuator exerts important forces and it can slightly move and bend causing disturbances. Nevertheless, the system is stable and can reach important levels of pressure with a rising time that is around 1 second in all 4 cases.

3.3.1 Preloading the brakes

DURING FINAL TESTING OF THE ROS NODE I FOUND OUT THAT THE PRELOADING OF THE BRAKES DOESN'T IMPROVE THE PERFORMANCE THAT MUCH BUT IF YOU DECIDE TO DO IT THEN IT SEEMS THAT IT WORKS BETTER IF YOU START FROM A LEVEL AROUND 0.5BAR (IN THIS SECTION I SUGGEST 0.05 WHICH SEEMS A BIT TOO LOW).. AND ALSO IT SEEMS THAT THE GAIN SCHEDULING IS CAUSING MORE PROBLEMS THAN IT SOLVES THEREFORE I JUST PUT THE PI-GAINS THE SAME IN THE END.. I THINK YOU'LL HAVE TO MAKE TESTS ON THE TRACK TO SEE HOW THE SYSTEM PERFORMS IN A DYNAMIC SITUATION AND EVENTUALLY RETUNE ALL THE PARAMETERS..

Reducing the controller gains to get rid of the overshooting described above, has the downside of making the system slow. In order to avoid overshoot but keep a quick braking time it was found out that instead of starting from no pressure on the brakes it was better to slightly increase

the level to approximately 0.1bar (0.05bar is not a recommended set-point because that's the threshold level where the PID gains are rescheduled and noise could make them jump between the two levels). At that level there is still no significant friction between the calipers and the discs of the brakes. Figure 3.11 shows the step responses from 0.1bar to 5, 10, 15 and 20bar. In all 4 cases the overshoot is much smaller or eliminated but the rise time is around 1 second.

For tasks like parking, where the driving program of the Ecocar should already know in advance that braking will be required, it will be important to preload the brakes with a pressure level around 0.1bar to get a faster response.

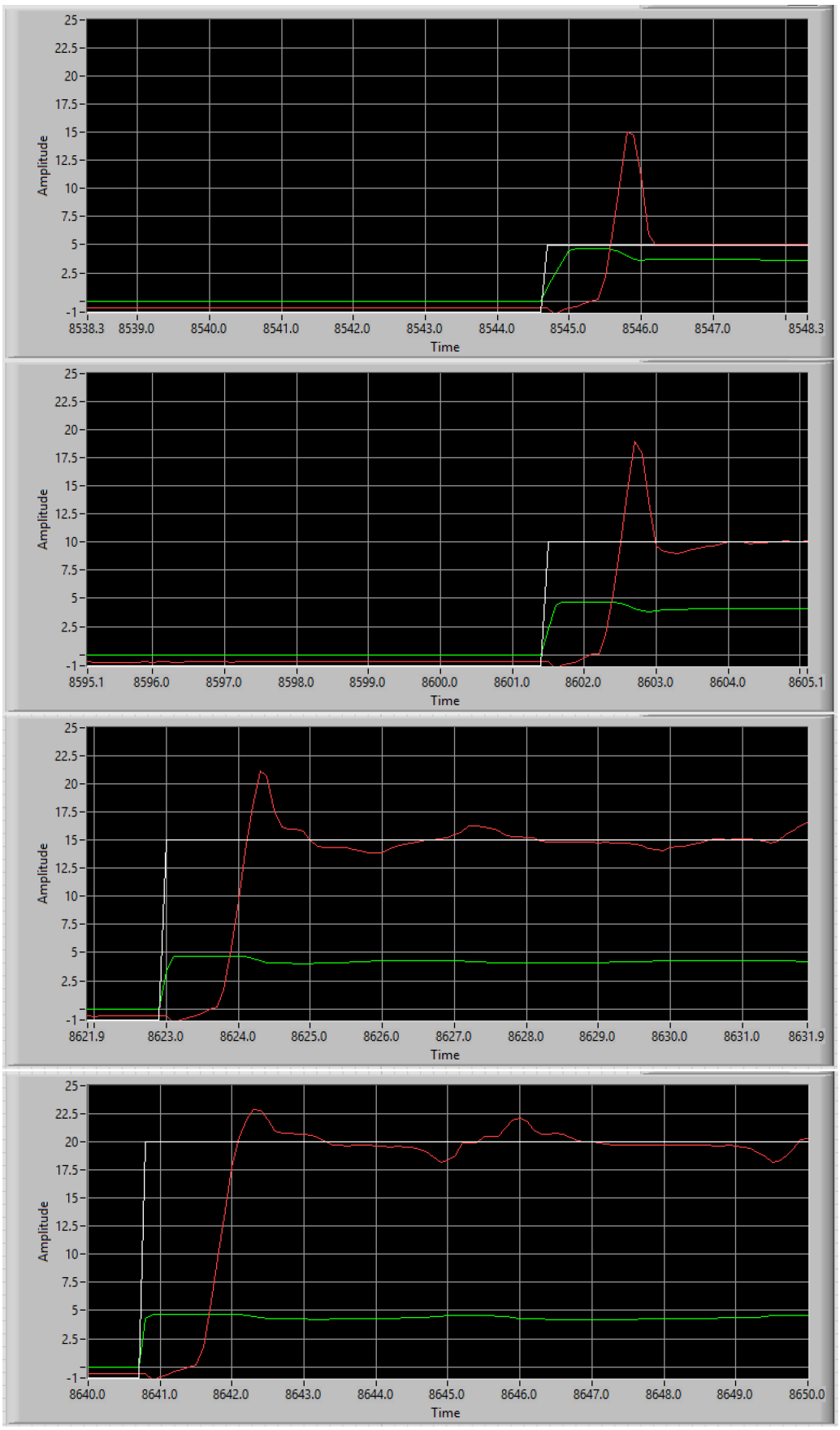

Figure 3.10: Step responses starting from no pressure to 5 (top), 10, 15, 20bar (bottom); set point white, output red, controller output [V] green; a significant overshoot can be seen

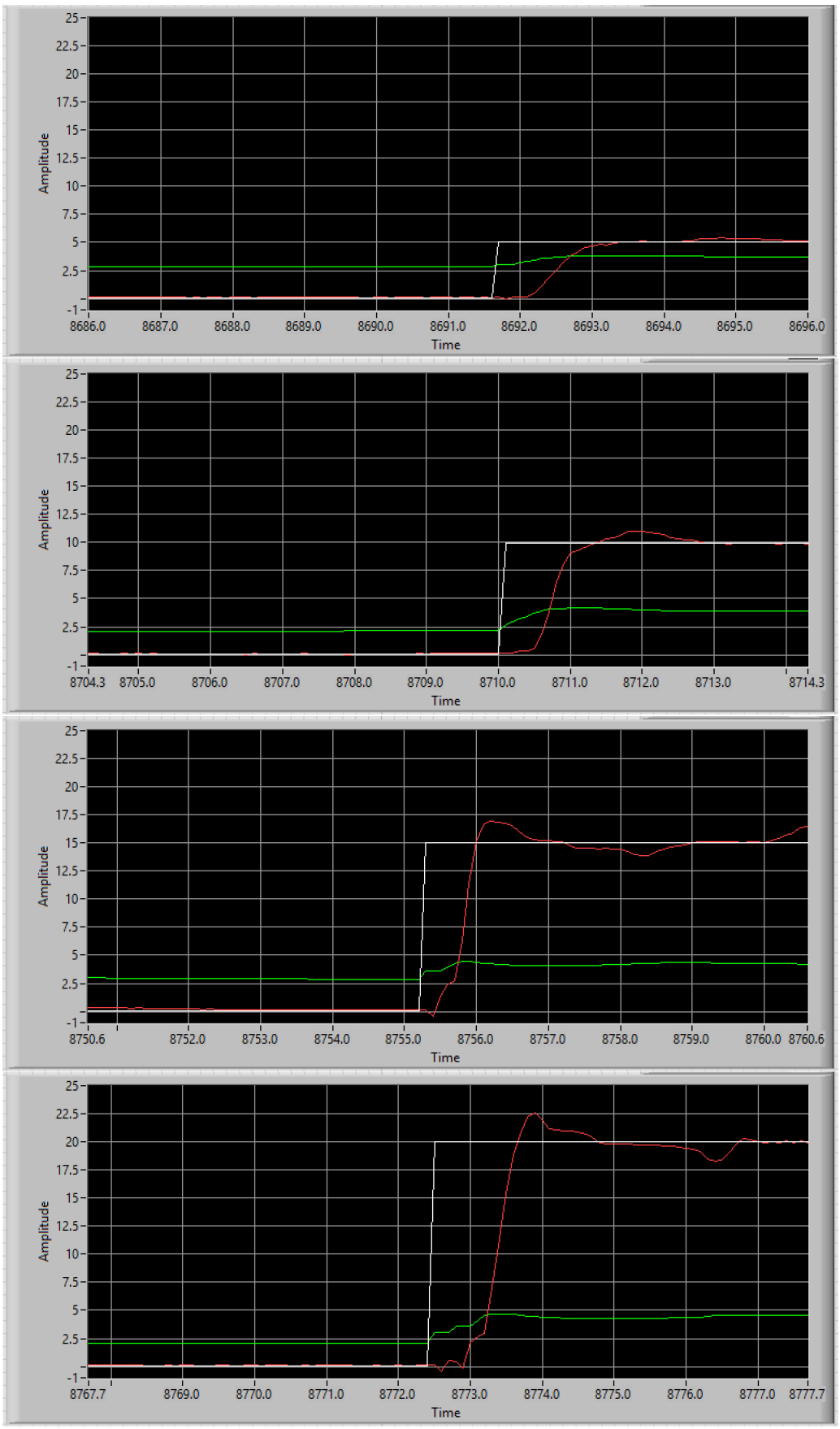

Figure 3.11: Step responses when preloading the brakes starting from 0.1bar to 5 (top), 10, 15, 20bar (bottom); set point white, output red, controller output [V] green; overshoot is very small or absent

3.3.2 Change to the mechanical system

During the testing phase of the controller it was found out that when using the front pressure sensor instead of the rear one the step response was significantly worse and even after a re-tuning of the control parameters it wasn't possible to get a satisfying pressure control. In other words, the dynamics of the front and rear brake system appeared different. It was found out that this was caused by the mechanical system that is supposed to balance out the force between the two brake cylinders (front and rear brake system). The mechanical team prepared a new prototype that significantly improved the equilibrium between the cylinders. The prototype was built with fiber wood and was not meant to handle high pressure levels. At the point in time when the report was written, the new mechanical system wasn't ready.

Chapter 4: Implementation of the pressure controller in ROS

Due to time constraints and low level of experience with ROS it was decided to modify the pressure controller node from a previous project instead of completely rewriting it. Although, the optimal solution would be to write a clean node from scratch.

4.1 Understanding the brake node

Different pressure sensor readings are published by the node *teensy_node* over the topic *teensy_read*. The ROS messages that contain the front and back pressure values can be accessed with the following commands:

```
rostopic echo /teensy_read/brake_pressure_1
rostopic echo /teensy_read/brake_pressure_2
```

It is possible to publish a message of type *dynamo_msgs* over the topic *cmd_brake_power* to the node *brake_node* with the following command

```
rostopic pub /cmd_brake_power dynamo_msgs/BrakeStepper "brake_stepper_engaged:
    true brake_power: 0.0"
```

This command allows to engage the brake controller and set a reference pressure.

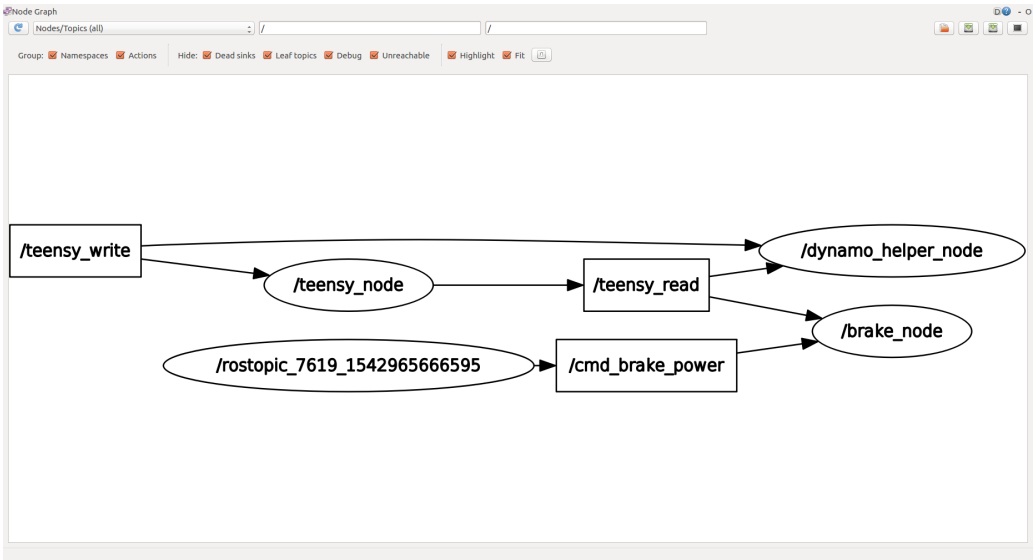

Figure 4.1: Diagram generated with rqt_graph: the node *brake_node* subscribes to the topics *teensy_read* and *cmd_brake_power*

4.2 Modifications to the brake node

The main code of the node is located in the folder .catkin_ws/src/brake/src/brake_node.cpp. The node itself wasn't modified but changes were made to the PID controller (BrakeSystem.cpp) and the header file (BrakeSystem.h) located in .catkin_ws/src/brake/src/brake/include.

In particular, the controller was modified in order to use aggressive PI-gains at low pressure level (see more in section 3.3) with a simple if/else statement:

```
if (press > Gain_sched)
{
        Kp = KC;
        Ki = IC;
}
else
{
        Kp = KC0;
        Ki = IC0;
}
```

LATER ON I FOUND OUT THAT THIS GAIN SCHEDULING IS NOT REALLY HELPING THAT MUCH SO YOU COULD ALSO JUST GET RID OF IT..
In addition, anti integral wind-up was added. In other words, the growth of the integral part of the control output is blocked when the actuator reaches its limits:

```
if (output > posMax || output < posMin)
{
        Iout = Ki * pre_integral; // Recalculate Iout with previous value if
            output is saturated
        integral = pre_integral; // Freeze integral
        output = Kout + Iout;
}
```

The controller gains and the threshold level for changing between the different PI gains is located in the header file:

```
#define KC 0.01 // This would be K_p in the parallel form of a PI --> G(s) = K_p
    + K_i/s
#define IC 0.2 // This is K_i
#define DC 0 // Disregarded --> PI only
#define KC0 0.1 //Gain scheduling: this is the more aggressive P-gain to be used
    only at low pressures
#define IC0 1 //Gain scheduling: this is the more aggressive I-gain to be used
    only at low pressures
#define Gain_sched 0.05 // Pressure level in bar where the controller gains
    change from KC0 to KC and from IC0 to IC
```

Furthermore, it was found out that the pressure value that is passed to the controller function is already in bar and therefore the measurement doesn't need to be scaled. By default the controller will use the pressure sensor of the rear wheels (number 2). However, if the front pressure sensor's value is at least 10% bigger, then its value will be passed to the controller. The rear pressure sensor is used as default because when the Ecocar is without top cover the front sensor's signal is not available.
I COULDN'T TEST THIS PART OF THE CODE IN PRACTICE BUT IT SHOULD WORK..

```
if (press1 > press2*1.1) // The 10% margin is to avoid jumping from one sensor
    to the other if values are close
{
        press = press1;
```

```
}
else
{
        press = press2;
}
```

Using the maximum between the two signals is an alternative option. However, it has the disadvantage that the system might jump from one sensor value to the other causing instability in the control loop if the front and the rear brake system present different dynamics).

When the set-point pressure is 0 bar the controller will go into open loop and set the output to 0V (fully retract the actuator). At the same time the integral action will be reset:

```
if(brakeSystem.power == 0)
{
    brakeSystem.SentToCon(brakeSystem.posMin);
        brakeSystem.integral = 0; // When the reference is 0 bar reset the
            integrator
        brakeSystem.pre_integral = 0;
}
```

At the point in time when the report was written, it wasn't possible to fully test the new braking system because the mechanical part still required an update (see section 3.3.2). However, the ROS pressure controller was tested just by pushing the pedal manually and the linear actuator was promptly responding, showing that the control loop was working correctly. However, it is important to note that the control parameters might need a re-tuning after the update of the mechanical system and whenever maintenance work is done on the brake system!

4.3 Testing the new system

IF YOU NEED TO TEST OR RETUNE THE PRESSURE CONTROLLER FOLLOW THIS STEPS

To test the system make sure that the brake node is running:

```
rosrun brake brake_node
```

In an other terminal the pressure signal can be plotted (pressure 1 might be added as well if available):

```
rqt_plot /teensy_read/brake_pressure_2
```

The axes can be adjusted in the rqt_plot interface by clicking "Edit curve lines and axes parameters". A good setting is x-axis from 0 to 10 seconds and y from 0 to maximum desired pressure. In another terminal the brake command can be launched. For example 10bar:

```
rostopic pub /cmd_brake_power dynamo_msgs/BrakeStepper "brake_stepper_engaged:
    true brake_power: 10.0"
```

At this point the actuator will start regulating (see figure 4.2).

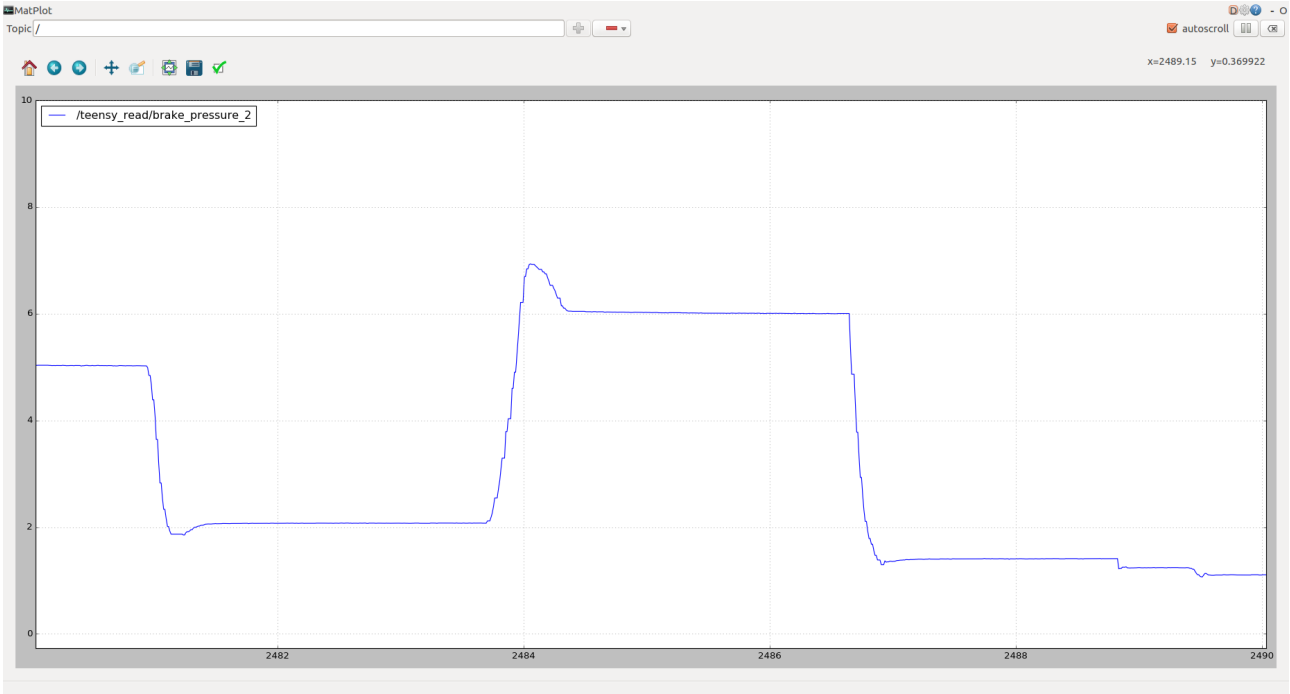

Figure 4.2: Step responses with rqt_plot

If you need to change the PI-parameters recompile everything with the command *catkin_make* (must be called from the folder catkin_ws).

Chapter 5: Conclusion

Thanks to the integration of a new controller for the linear actuator it was possible to significantly improve the performance of the autonomous braking system.

Several tests done during the prototyping phase made it possible to individuate weak points in the mechanical part of the brake system and help the mechanical team to improve it.

Modifications were done to the ROS brake node and the implementation of a digital pressure controller is finally working.

Bibliography

[1] Jrk G2 Motor Controller User's Guide - Software,
https://www.pololu.com/docs/0J73/3