Using\_types.h:

All special variable types like elapsed time or number of steps for pid optimizer are stored here to be accessible for everybody, who may need it to deliver some variables from his class to another.

Pid.h:

There is a Pid.h file, which contains a struct with all the PID settings.

These are the fields ***kp***, ***ki***, ***kd*** and ***error***.

The part of the ***Pid struct*** called ***error*** is used by the Pid\_optimizer to be able to compare different Pid controller settings and in after PID tuning, the relative error is stored in the error field of the Pid struct.

Ball:

Its current position and velocity can be set or asked and there is a function ***push(Push::left/right, velocity)*** to be pushed.

Beam:

It tries to reach the desired angle, but the movements are limited in angle and angular velocity. Therefor the beam has to know, how much time expired since the last call. It throws an exception, if the elapsed time or if the elapsed time is not larger than null. The elapsed time is given to this class in milliseconds.

The beam’s length can be asked via ***get\_length()*** and also the angle via ***get\_angle()***. The beam’s desired angle can be set via ***set\_angle(angle, elapsed\_time)***.

Pid\_optimizer:

It tries the desired settings for trying different PID parameters. These settings are stored in the struct ***Opt\_param*** and contain a starting value for kp, ki and kd and the number of iterations. In each iteration the kp, ki and kd settings are first increased by half of the value in the last iteration and if it does not make the controller controlling faster and more precise, kp, ki or kd are decreased at the same value again.

The optimizer has a an ***actual\_pid\_*** struct and an ***optimal\_pid\_*** struct. When the Pid\_optimizer has finished working, optimal\_pid\_ struct is updated.

Pid\_optimizer also has a function for trying new kp, ki and kd settings; one function for every parameter.

The function called ***watch\_error(controller&)*** is an internal function, which may only be called by the optimizer itself. This function is used to let the controller simulate a ball on a beam it has to control and monitor the error while the ball is moving and tried to be balanced. After a specified time, the simulation is aborted and the sum of all position errors during simulation relativized over the initial maximum error (if the ball would be positioned on different positions on the beam, this would be essential) and the simulated time.

The Pid\_optimizer can be accessed via its function ***get\_optimum(optimizer\_parameters&)***. The user just has to call this function and this function is calling the internal function ***run(optimizer\_parameters&)*** and is returning the optimal PID settings afterwards.

The internal function ***run(optimizer\_parameters&)*** is creating an object ball, beam and controller. With these objects the controller is getting optimized.

*BalanceOptimizer:*

In this program an optimizer is created, which is initialised with default optimizer parameters and once the optimizer finished running, the optimal PID settings are written out to console.

Controller:

Is designed to control the angle of the beam in such a way that the ball reaches the desired position. The control() method implements a PID controller that outputs an angle the beam should reach. Another named update() updates the position of the beam and the ball as well as calculates the current velocity of the ball. These methods are given the time since their last call so they can calculate the needed parameters.

Clock:

Simply an object to keep track of the passed time and call the current system time.

Problems:

We tried saving the time since the last call of the methods update() and control() in a static variable. However, we kept getting an exception thrown at our faces when writing our unit test and couldn’t figure out why. Other than that the program should function just fine.