**USE-Case**

Das Zusammenspiel der einzelnen Komponenten der Applikation wird im Folgenden anhand eines Beispiels erklärt.

Beim Programmstart werden durch das Model drei \textit{closedLoops}

(geschlossenen Regelkreise) für die Phasengang-Methode sowie vier weitere für die Faustformeln erzeugt. Jeder \textit{closedLoops} ist von Beginn an einem Berechnungstyp (Phasengang-Methode, Faustformel) zugewiesen. Er ist bereit, Daten aufzunehmen und zu verarbeiten.

Über die drei Eingabefelder $K\_s$, $T\_u$ und $T\_g$ werden die Werte der vermessenen Regelstrecke durch den Benutzer eingegeben. Durch Drücken des Buttons „Berechnen“ werden die Eingaben durch den \textit{GUIController} auf Zulässigkeit überprüft. Erfüllen sie die erforderlichen Kriterien nicht, wird eine Benachrichtigung mit Hinweis auf den Fehler oberhalb des Buttons ausgegeben und die Berechnung nicht ausgelöst.

Haben die Eingaben die Überprüfung durch den \textit{GUIController} bestanden, fragt dieser zusätzlich die aktuellen Werte/Zustände der Slider für Überschwingen und Optimierung sowie den Reglertyp auf dem GUI ab und leitet alle Daten mittels \textit{setData()} an das Model weiter. Dieses erzeugt einen Path (Strecke) aus den Eingabewerten. Das Model errechnet den Optimierungs-Offset und weist die Daten den entsprechenden \textit{closedLoops} der Phasengang-Methode sowie der Faustformeln mittels der\textit{setData()} Methode zu. Jeder \textit{closedLoops} leitet die Daten an den zugehörigen Controller weiter, der die Reglerwerte berechnet. Zu Beginn betrachten wir den Regler nach Oppelt genauer.

Die Klasse \textit{Oppelt} erbt von der abstrakten Klasse Controller und besitzt somit alle setter- und getter-Methoden der Oberklasse. Als Input stehen die Informationen der Strecke sowie der Reglertyp (PI, PID) zur Verfügung. Je nach gewähltem Berechnungstyp werden die Reglerwerte „reglerkonform“ berechnet und gespeichert. Weiter werden die Werte in die „bodekonforme“ Darstellung umgerechnet und ebenfalls abgespeichert.

Die Berechnungstypen Rosenberg, Chien/Hrones/Reswick (20%) sowie Chien/Hrones/Reswick (aperiod.) funktionieren analog dem Berechnungstyp Oppelt und werden nicht weiter ausgeführt.

Ein spezielles Augenmerk richten wir nun auf die Berechnung der Phasengang-Methode. Auch die Klasse \textit{phaseResponseMethod } erbt von der Klasse Controller und bringt die bereits erwähnten setter- und getter-Methoden mit. Die Input-Werte sind analog derjenigen der Faustformeln. Zusätzlich werden die Informationen zum Überschwingen, der Optimierung sowie Tp in die Berechnung mit einbezogen und die gesetzten Input-Werte werden den lokalen Attributen zugewiesen. calculateOvershoot() wird ausgelöst und setzt das Attribut phiU, welches für das korrekte Überschwingen benötigt wird. Darauf folgend wird calculate() aufgerufen. Diese Methode berechnet anhand der Methode createOmegaAxis() die diskrete Omega-Achse in Abhängigkeit der Zeitkonstante der Regelstrecke. Die Übertragungsfunktion in s der Regelstrecke wird für alle Punkte von Omega berechnet. calculateTnk() wird ausgelöst. Diese Methode berechnet Tnk und Tvk unter Zuhilfenahme der diskreten Werte nach dem Prinzip der Phasengang-Methode. Daraus resultiert die Übertragungs-Funktion des Reglers. Falls Tp = 0 aus der Eingabe übergeben wurde, so wird an dieser Stelle Tp berechnet. Krk wird gemäss der Phasengang-Methode mittels calculateKrk() berechnet. Zudem beinhaltet. die Methode calculateKrk() den Aufruf von calculateControllerConf() und setUTF(). calculateControllerConf transformiert die Werte in die „reglerkonforme“ Darstellung. setUTF() setzt die Übertragungs-Funktion des Reglers.

Im closedLoop wird nun die Methode calculate() ausgeführt, welche mittels calculateStepResponse() die Schrittanwort des geschlossenen Regelkreises berechnet. Falls es sich bei der Berechnung des Reglers um die Phasengang-Methode handelt, wird zusätzlich die overShootOptimization() aufgerufen. Diese Methode ändert den Wert von Krk so lange, bis das gewünschte Überschwingen erreicht wird.

Sobald die Berechnungen aller \textit{closedLoops}abgeschlossen sind, wird im Model \textit{notifyObserver()} ausgelöst. Hiermit wird die View darüber informiert, dass Änderungen im Model vorgenommen wurden und die Methode \textit{update()} in den jeweiligen View-Unter-Klassen aufgerufen werden soll. Somit aktualisiert sich die View und die neu berechneten Regler-Werte sowie die Strecken-Ordnung werden ausgegeben und die Plot-Daten werden aktualisiert. Im Speziellen für die Phasengang-Methode, werden auch Start-Werte für $T\_p$ gesetzt.

Der Benutzer hat nun die Möglichkeit die Resultate der Phasengang-Methode weiter zu optimieren. Über das Panel \textit{Optimierungen} stehen die Slider ’’Überschwingen“ sowie ’’Optimierung“ zur Verfügung. Das Überschwingen kann in vorgegebenen Schritten in Prozenten bestimmt werden. Die Optimierung schiebt den Phasenrand bzw. den Regler-Knickpunkt in die positive sowie negative Richtung zugleich, und wird mittels zwei separaten Plots dargestellt. Weiter können die Werte für $T\_p$ nachträglich für jede der drei kurven individuell angepasst werden.   
Sobald einer der drei Parameter (Überschwingen, Optimierung, $T\_p$) verändert wird, wird über den \textit{GUIController} die jeweilige setter-Methode im Model aufgerufen. Das Model gibt die Daten an den jeweiligen \textit{closedLoop} weiter, der diese wiederum den Methoden der Phasengang-Methode weiterleitet. Sobald die neu berechneten Werte vorliegen wird \textit{notifyObservers()} aufgerufen und die View aktualisiert.

Die Plots sowie die Optimierungs-Schaltflächen sind nur dann sichtbar, wenn die CheckBox ’’Erweiter“ aktiviert ist. Durch deaktivieren dieser CheckBox, kann das Programm in einer Klein-Ansicht, ohne grafische Ausgabe, bedient werden. Die einzelnen Plots können über CheckBoxen unterhalb des Plot-Bereichs dazu- oder weggeschaltet werden.

Über den Button „Löschen“ können alle Regler- sowie Plot-Daten gelöscht werden.