Real-Time Systems

Laboratory 1: Control of the Ball and Beam Process

2021 Version

Preparatory Exercise

In order to be able to complete this laboratory, you must have finished Exercise 3. This will not be monitored. However, we strongly recommend it.

Code Skeleton

To simplify the coding process, we have provided a code skeleton, including all the java files you will need. It can be downloaded on Canvas.

Introduction

Your task is to write a Java program that controls the ball and beam process. The Java program should consist of two parts:

- `OpCom`, a version of the Swing-based GUI from Exercise 4. Provided to you.
- `Regul` from Exercise 3, structured as `BallAndBeamRegul`, but with the public interface provided to you.

The main difference between the new Regul class and the one used in Exercise 3 is that the GUIs for the inner PI controller and the outer PID controller are now implemented by `OpCom` rather than by internal GUI classes. Thus, it is **important** that you **do not** include the `PIGUI` and `PIDGUI` classes from Excerise 3 and that you use the `OpCom` class we are providing you here.

A predefined `ReferenceGenerator` class is used to provide the reference signal. This class has its own GUI. From the GUI you can decide to either use a squarewave signal as the reference or to manually set the reference using a `JSlider`.

Simulator

Compared to Exercise 3, you will now use a more advanced simulator, rather than than the simple virtual simulator we used in Exercise 3. The advanced simulator uses the following public interface:

```
  class BallBeamAnimator {
    // Gets the beam angle
    public double getBeamAngle();

    // Gets the ball position
    public double getBallPos();

    // Actuates control signal to motor
    public void setControlSignal(double u);
}
```

JavaFX

Important to note is that the simulator uses JavaFX to animate the process. JavaFX is included in all Java version prior to (and including) Java 8. This means that if you wish, you should be able to use the VirtualBox provided to you under the *Additional Exercise Material* page in Canvas.

If you wish to run a later version of Java on your own machine, you will have to download an external library for JavaFX. Instructions for different operating systems follow:

1. **Windows:**
    1. Download "JavaFX Windows SDK" (version 11.0.2) from the Gluon Website.
    2. Move the .zip file to your code repo.
    3. Unzip the .zip file.
    4. When compiling and executing your files, include the following classpath:

       ```
       javac --module-path javafx-sdk-11.0.2/lib --add-modules javafx.base,javafx.graphics,javafx.swing -cp .;regler.jar *.java
       java --module-path javafx-sdk-11.0.2/lib --add-modules javafx.base,javafx.graphics,javafx.swing -cp .;regler.jar Main
       ```

       NOTE that regler.jar also needs to be included.
2. **Mac:**
    1. Download "JavaFX Mac OS X SDK" (version 11.0.2) from the Gluon Website.
    2. Move the .zip file to your code repo.
    3. Unzip the .zip file.
    4. Move the `javafx-sdk-11.0.2/lib/` folder to your code folder.
    5. When compiling and executing your files, include the following classpath:

       ```
       javac -cp .:regler.jar:lib/javafx.graphics.jar:lib/javafx.base.jar:lib/javafx.swing.jar:lib/ *.java
       java -cp .:regler.jar:lib/javafx.graphics.jar:lib/javafx.base.jar:lib/javafx.swing.jar:lib/ Main
       ```

       NOTE that regler.jar also needs to be included.
3. **Linux:**
    1. Download "JavaFX Linux SDK" (version 11.0.2) from the Gluon Website.
    2. Move the .zip file to your code repo.
    3. Unzip the .zip file.
    4. Move the `javafx-sdk-11.0.2/lib/` folder to your code folder.
    5. When compiling and executing your files, include the following classpath:

```
javac -cp .:regler.jar:lib/javafx.graphics.jar:lib/javafx.base.jar:lib/javafx.swing.jar:lib/ *.java
java -cp .:regler.jar:lib/javafx.graphics.jar:lib/javafx.base.jar:lib/javafx.swing.jar:lib/ Main
```

NOTE that regler.jar also needs to be included.

Troubleshooting

Mac: It is very likely that the first time you will try to execute the JavaFX files on macOS it will prevent that because it doesen't recognize the signature of the developer. A window will be displayed showing a message like: *"SOME-FILE" cannot be opened because the developer cannot be verified* . We need to tell the OS that it is safe to execute these files. To do so you will have to iterate the following steps a couple of times:

- Try to execute the compiled files with the `java` command and get the error message.
- Answer *Cancel* to the error message.
- Go to *System Preferences -> Security & Privacy* and click *Allow Anyway*.
- Now another type of error will appear but this time it should give you the option *Open Anyway*, click it.
- This has to be done for different files, hence some iterations (3 to 5) will be needed. But once it is set up you will not need to do it again.

Linux: If the following warning message comes up during Execution

```
WARNING: System can't support ConditionalFeature.SCENE3D
```

try using the following compilation and execution commands instead:

```
javac -cp .:regler.jar:lib/javafx.graphics.jar:lib/javafx.base.jar:lib/javafx.swing.jar:lib/ *.java
java -cp .:regler.jar:lib/javafx.graphics.jar:lib/javafx.base.jar:lib/javafx.swing.jar:lib/ -Dprism.forceGPU=true Main
```

ModeMonitor

The public interface of `ModeMonitor` is the following:

```
  public class ModeMonitor {
  private Mode mode = Mode.OFF; // Off mode to start with

  // Sets new mode
  public synchronized void setMode(Mode newMode) {
    mode = newMode;
  }

  // Returns the current mode
  public synchronized Mode getMode() {
    return mode;
  }

  // Existing modes
  public enum Mode {
    OFF, BEAM, BALL;
  }
}
```

This class is here to make the shared resource `Mode` mutually exclusive.

OpCom

The public interface of `OpCom` is the following:

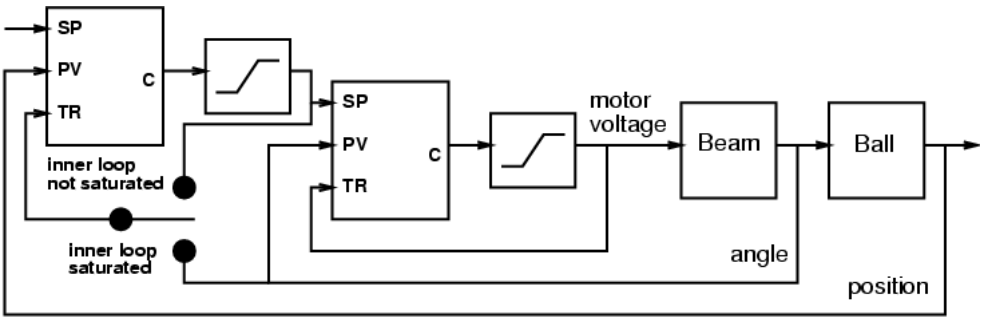
```
  // Constructor. Note: Different from Exercise 4.
  public OpCom(int plotterPriority, ModeMonitor modeMon);

  // Passes in a reference to Regul. Called from Main.
  public void setRegul(Regul r);

  // Build up the GUI. Called from Main.
  public void initializeGUI();

  // Starts the plotting within OpCom. Called from Main.
  public void start();

  // Plots a new control signal data point
  public void putControlData(double t, double u);

  // Plots a new measurement data point
  public void putMeasurementDataPoint(double t, double yRef, double y);
```

The control modes are acquired through the `ModeMonitor` class.

Regul

The `Regul` class should contain one PI controller for the inner loop and one PID controller for the outer loop. Use a single thread for the execution of both the controllers in the same way as in the exercise. `Regul` receives its reference signal by calling the `getRef()` method of the `ReferenceGenerator`.

The controller should use the cascade structure according to the figure below. The code should be written so that the **delay** between the sampling of the measurement signals and the generation of the control signal is **minimized**.



`Regul` should run in one out of three different modes: OFF, BEAM, and BALL (the mode is kept by the `ModeMonitor`). In each loop, the regul thread should call `sendDataToOpCom()` to send the control and measurement data point to the OpCom. If the mode is OFF the signals sent to `OpCom` (y, yref and u) should all have the value 0.

A difference compared to Exercise 3 is that the reference signal now is plotted by the GUI rather than by the virtual simulator. Hence, the reference signal should no longer be output using an analog output. Instead, the reference signal should be sent to `OpCom` by calling `sendDataToOpCom()`.

Regul Public Interface

The Regul class should have the following public methods:

```
// Constructor
public Regul(int priority, ModeMonitor modeMon);

// Passes in a reference to OpCom. Called from Main.
public void setOpCom(OpCom o);

// Passes in a reference to ReferenceGenerator. Called from Main.
public void setRefGen(ReferenceGenerator r);

// Sends the data to OpCom
public void sendDataToOpCom(double yRef, double y, double u);

// Set the parameters of the inner PI controller (inner controller).
// The PIParameters class is the same as in Exercise 3 (without
PIGUI).
// Not synchronized, the synchronization is handled by PI.
public void setInnerParameters(PIParameters p);

// Returns the current parameters in the inner loop.
// Only called during OpCom GUI initialization.
// Not synchronized, the synchronization is handled by PI.
public PIParameters getInnerParameters();

// Set the parameters of the outer PID controller (outer controller).
// The PIDParameters class is the same as in Exercise 3.
// Not synchronized, the synchronization is handled by PID.
public void setOuterParameters(PIDParameters p);

// Returns the current parameters in the outer loop.
// Only called during OpCom GUI initialization.
// Not synchronized, the synchronization is handled by PID.
public PIDParameters getOuterParameters();

// Called from OpCom when the Stop button has been pressed,
// before the system is shut down.
public void shutDown();
```

PI and PID

Use your PI and PID classes (removing the `PIGUI` and `PIDGUI` dependence) from Exercise 3 with the following extensions:

```
// PI

// Sets the I-part of the controller to 0.
// For example needed when changing controller mode.
public synchronized void reset();

// Returns the current PIParameters.
public synchronized PIParameters getParameters();



// PID

// Sets the I-part and D-part of the controller to 0.
// For example needed when changing controller mode.
public synchronized void reset();

// Returns the current PIDParameters.
public synchronized PIDParameters getParameters();
```

Main

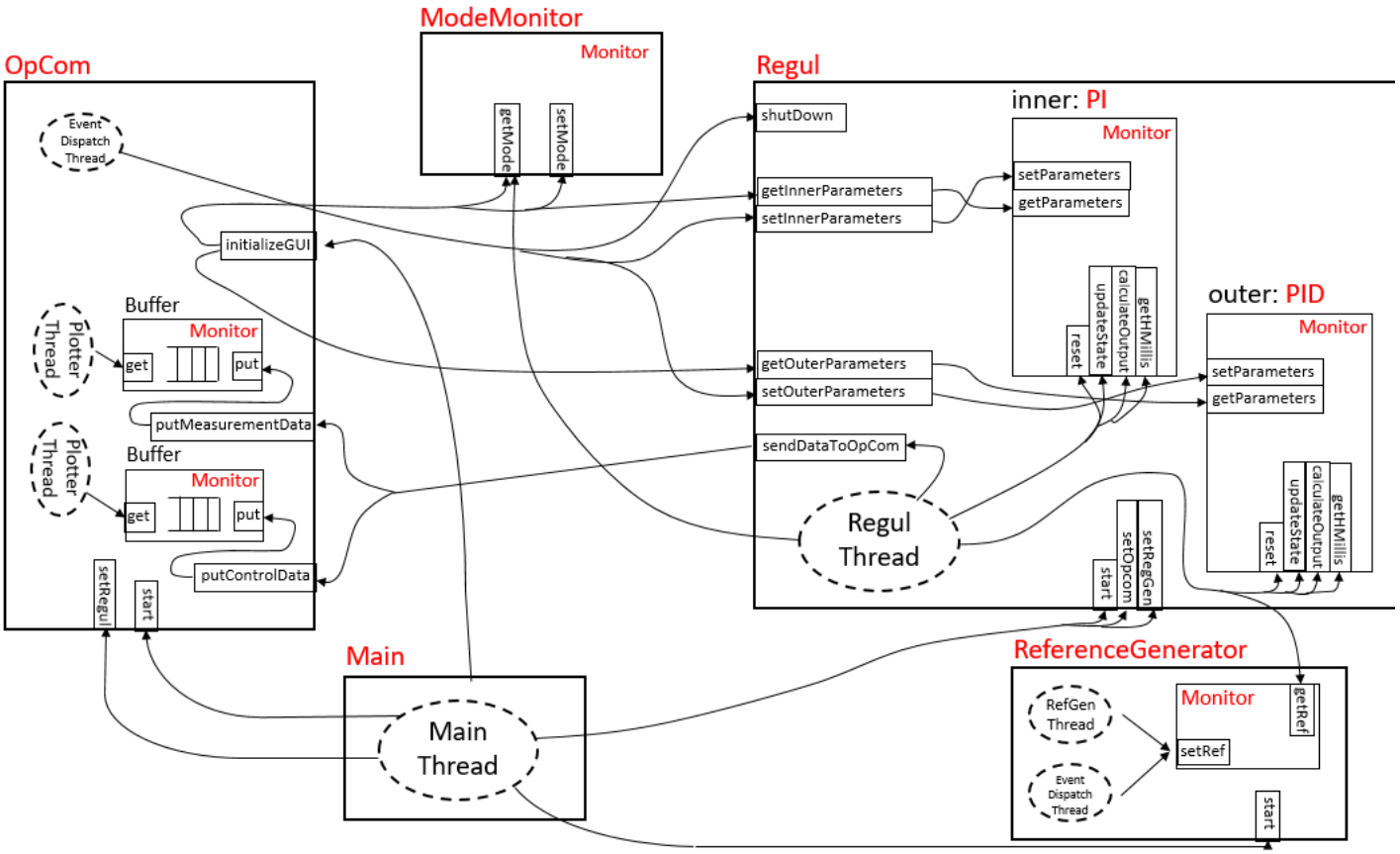
The following Main class should be used.

```
import javax.swing.*;

public class Main {
  public static void main(String[] argv) {
```

```
    // Initialise ModeMonitor
    ModeMonitor modeMon = new ModeMonitor();

    // Thread priorities
    final int regulPriority   = ...;
    final int refGenPriority  = ...;
    final int plotterPriority = ...;

    // Initialise Control system objects
    ReferenceGenerator refGen = new ReferenceGenerator(refGenPriority);
    Regul regul = new Regul(regulPriority, modeMon);
    final OpCom opCom = new OpCom(plotterPriority, modeMon);

    // Setting dependencies
    regul.setOpCom(opCom);
    regul.setRefGen(refGen);
    opcom.setRegul(regul);

    // Start threads
    Runnable initializeGUI = new Runnable(){
      public void run(){
        opCom.initializeGUI();
        opCom.start();
      }
    };
    try{
      SwingUtilities.invokeAndWait(initializeGUI);
    }catch(Exception e){
      return;
    }

    refGen.start();
    regul.start();
  }
}
```

Structure Diagram

The design can be summarized in the figure below. The notation introduced in the Buttons exercise is used.



NOTE: In the diagram, the PI and PID classes are drawn as if they are inner classes of Regul but they should be kept as ordinary, "top-level", classes.

---

1. Update your PI and PID classes using the modified interface.
2. Implement `Regul` based on your `BeamRegul` and `BeamAndBallRegul` classes from Exercise 3. It should use the `BallBeamAnimator` simulator. Make sure that it sends plot data to `OpCom` in the correct way. We have provided a code skeleton as a starting point.

Make sure that:

1. The I/O delay, that is the delay between sampling of the measurement signals and the generation of the control signal, is minimized.
2. The lock times aren't longer than necessary.
3. The outer loop tracking is in accordance with the structure figure above.

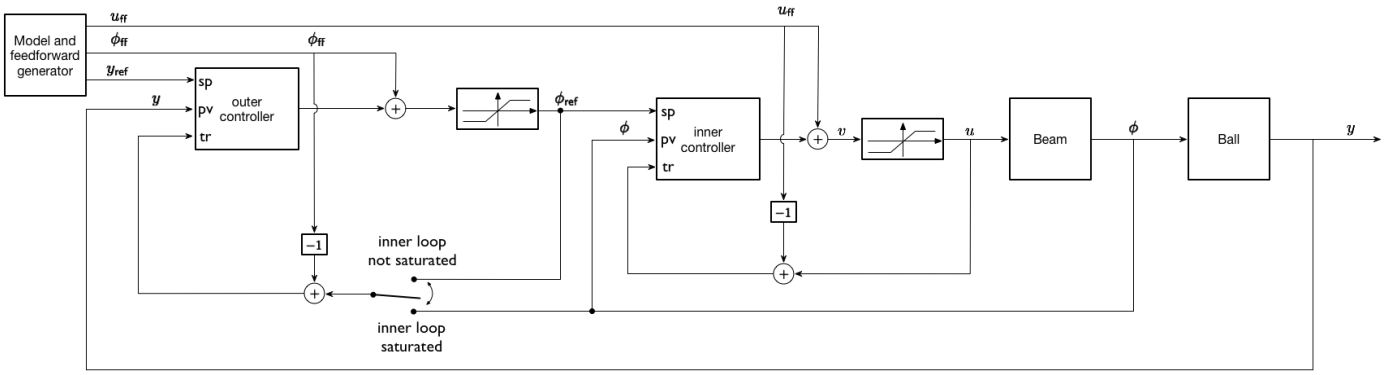
3. Test your program:

- Start by ensuring that your program work in the BEAM mode. Start with the controller parameter values that you used for Excercise 3.
- Once your program works in BEAM mode, go on to BALL mode. Start with the controller parameter values that you used for Excercise 3.

Time-Optimal Feedforward Generator

In the next part you will extend the previous control-structure with a time-optimal feedforward generator. You have already been provided all the necessary classes for this, and will only have to do some minor changes in order to make it work. Before moving on, you should recall the lecture slides from *Lecture 10*.

Below you can see a block diagram for the new control structure. To change your current control-structure into this, you will need to call two new methods from the reference generator in order to get the feedforward terms: `referenceGenerator.getPhiff()` and `referenceGenerator.getUff()`. You should also recall from the lecture that when using the feedforward generator you will need to use gamma=1 when computing the D-term in the PID-controller. Therefore, in `PID.java` you will have to change `D = ad*D - bd*(y - yOld)` to `D = ad*D + bd*(e - eOld)` (NOTE the plus-sign!)



1. Update the `Regul.java` to use the new feedforward signals
   - The outer loop can use the method `getPhiff()` to retrieve the feedforward term
   - The inner loop can use `getUff()`
   - When implementing the anti-windup tracking, you should use `updateState(u-uff)` for the inner loop, and `updateState(angRef-phiff)` for the outer loop (or `updateState(ang-phiff)` if the inner loop is saturated)
2. Update `PID.java` to use gamma = 1:
   `D = ad*D - bd*(y - yOld)` should be changed into
   `D = ad*D + bd*(e - eOld)`
3. Compile and test the new system using time-optimal feedforward generator. Remember to switch the reference generator into "Time-optimal mode". Do you see any difference?

Analysis

1. Change the sample time of the system. What happens with the system dynamics when the sample time becomes very small and very large?
2. Give *at least* three shared resources that we have to handle with extra care.
3. Why do we put the `synchronized` keyword on every method in the `PI` and `PID` classes? Motivate.
4. Why would it be poor design to add an integrator to the inner loop controller?
5. In the course so far we usually sleep threads using the following code snippet:

```
t += h; // t was the previous release time and h is sample time
duration = t - system.currentTimeMillis();
if (duration > 0) {
  try {
    sleep(duration);
  } catch (InterruptedException x) {
    // Do something
  }
}
```

   Explain what it means for the controller that `duration` is less than or equal to zero.
6. Briefly describe why we are using feedforward.

Submission Instructions

In order to pass Lab 1 you will have to submit:

1. The answer to the *Analysis* exercises above (preferably in a .txt, .pdf, or .md file),
2. Your code fulfilling the following requirements:
   - Minimising delay,
   - Minimising lock time,
   - Following the reference signal closely (particularly in time-optimal mode with feedforward),
   - Providing a reasonable plant behaviour and control signal.

The assignment (text document and code) should be submitted in a compressed format (preferably as a .zip file).