The main objective of this work is to design a controller for speed and position control.

Step #1- Speed Control

1. Consider the simulation of a motor (EMG30 motor) and associated load, available in the SiFEUP course contents for SimTwo simulator. Download the file "SimTwo2020.zip". Unzip the file and start the simulator "SimTwo2020.exe".

a. Manually control the motor by varying the voltage applied (form sheet, Ctr+H). Record the pulses generated by the encoder for each control period for different voltage values. Apply voltage values from 0,1 Volts to 15 Volts. Detect the dead-zone and saturation values. Register the maximum speed that the motor can achieve (Wmax).

b. Click in the scene image and go to the code editor (Ctr+E). In the procedure Control implement a proportional speed controller. Change the controller gain and observe the different responses to step variations at the reference value.

```
%The output sensor is an encoder. You read the impulses trough the instruction:
pulses := GetAxisOdo(iRobot, 0)

%You can convert from Pulses per Control Period to rad/s trough:
PulsesToSpeed :=  2 * PI / (1360 * Ts);
Speed := Pulses * PulsesToSpeed;

//And it is necessary an integration to get the position value:
Pos := Pos + Speed * Ts;

//Show the current position and speed at the sheet form (Ctr + H):
SetRCValue(3,3,format('Pos: %.2g',[Pos]));
SetRCValue(3,2,format('%.2g',[Speed]));

//Read from the sheet the reference and PID gain:
SpeedRef := GetRCValue(15, 4);
Kc := GetRCValue(15, 7);

//Calculate the control variable (Volts) and send it to the plant (motor+…):
Error := SpeedRef – Speed;
Vm := Kc*Error;
SetRCValue(1,2,format('%g',[Vm]));
SetAxisVoltageRef(iRobot, iAxis, Vm);
```

In the sheet form, set the reference to around 20% of Wmax and then to 90% of Wmax and the gain to values between 0.05 and 5.

Compile and run the program with F9. Observe the signal in the chart form (Ctr+T, select Speed only, at the bottom deselect Freeze and click on Refresh).
Don't copy-paste from the pdf to the editor. Hidden characters are copied and the program don´t run.

c. Note the instability of the system to high gains. What is the reason for this result?

d. Record the error in steady state and observe its evolution with the gain variation.

e. Implement a Proportional and Integral controller. The integral of the variable Error is numerically approximated by: S_error := S_error + error * Ts. The variable S_error must be global (top of the text code), type double and must be initialized at zero in procedure Initialize. In the beginning, use a low gain in the integral part. Start with an Integral Time of 5 seconds and go to lower values.

Step #2 – Plant model and Sped controller tunning

2. Find an approximate model of the motor plus load (input voltage. output speed), applying a step far from the dead-zone and saturation (for example from 2 Volts to 10 Volts) and recording the evolution of speed.
Use the "Freeze" checkbox to freeze a step response in the Graph form and the "Save Log" button to create a text file at the configuration sub-directory (MotorModel_V2) of Simtwo with the graph values.
Import these file to Excel, Matlab or another similar program and calculate the time constant.

3. With the above model calibrate the PI controller using the IMC method with a time constant in closed loop 0,1seg and 0,15seg. For each case record the step response and the time constant value in closed loop actually obtained.

4. Comment the relationship between the time constant value in closed loop and the sampling period. Do you think it is appropriate? What is the reason, in this case, that makes it impossible to use a lower value for the sampling period?

5. Implement a filter for the reference signal to deal with the overshoot in a step input. The pole of the filter must cancel the zero in the closed loop transfer function introduced by the PI controller. The filter must have also unitary gain at low frequencies.

6. Tune the PI speed controller using the Bessel Prototype with a settling time equal to 0.4 sec.

Step #3 – Position Control

7. Try to implement a PD position controller. Calculate the closed-loop transfer function and use a Bessel prototype with 0,4 seconds settling time. Repeat with a settling time equal to 0.8 seconds.

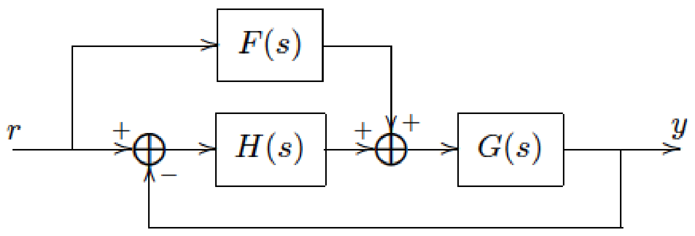
Calculate the position using the expression "Pos := Pos + PulsesToPos*Pulses".

8. Repeat now with a cascade controller.

9. To avoid the limit cycle oscillation in the cascade controller, introduce a condition that sets to zero the value for the velocity reference if the position reference is constant and the position error is smaller than a threshold value.

Step #4 – Controllers improvement with feed forward signal

10. Try to improve the speed controller with a feed forward signal. Change the reference signal to a sinusoid and test it with different frequencies. Adjust the amplitude and frequency avoiding always the saturation in the control signal.



11. Try also to improve the position controller with a feed forward signal. Test cascade and non-cascade versions.
12. Test the robustness off the position controllers to a wrong process model. Suppose that the gain can be 50% lower and the constant time 50% higher than the nominal value.

Step #5 – Generating Reference signals with controlled derivative

13. Implement a "square shaped" reference signal using the Cubic Hermite spline for the transitions. Adjust the amplitude and frequency avoiding always the saturation in the control signal. Test the controllers with that new reference signal.

How to create a LOG FILE

```
var
  Log: TStringList;
  LogOn: boolean;


procedure Control;
...
 if LogOn then begin
   // x y theta enc1 enc2 Laser
   txt := format('%d; %g; %g;  %g; %g; %g; %g; %g; %g; %g; %g', [nBeacon, dBeacon,
NormalizeAngle(Atan2(BeaconPos[nBeacon].y-y, BeaconPos[nBeacon].x-x) - theta),
BeaconCluster[nBeacon].dist, BeaconCluster[nBeacon].ang, x, y, theta,
GetRobotX(0), GetRobotY(0), GetRobotTheta(0)]);
   log.add(txt);
 end;

 if RCButtonPressed(2, 9) then begin // Start Log
   if log <> nil then Log.free;
   Log := TStringList.create;
   LogOn := true;
   SetRCValue(1,9,'Log ON');
 end;

 if RCButtonPressed(2, 10) then begin // Stop Log
   LogOn := false;
   SetRCValue(1,9,'Log OFF');
 end;

 if RCButtonPressed(2, 11) then begin // Save Log
   if log <> nil then Log.savetofile(GetRCText(2, 12)+'.csv');
 end;
```

```
...

procedure Initialize;

  Log := nil;
  LogOn := false;
  SetRCValue(1,9,'Log OFF'
```

How to create an USER CHART:

```
var
  ChartOn: boolean;


procedure Control;
...
  if RCButtonPressed(5, 9) then begin // Chart ON
    if ChartOn = TRUE then begin
      ChartOn := FALSE;
      SetRCValue(4,9,'Chart OFF');
    end else begin
      ChartOn := TRUE;
      SetRCValue(4,9,'Chart ON');
    end;
  end;
  if RCButtonPressed(5, 10) then begin // Clear Chart
    ChartSeriesClear(0, 0);
    ChartSeriesClear(0, 1);
  end;
  if chartON then begin
    ChartSeriesAddXY(0, 0, t, PosRef);
    ChartSeriesAddXY(0, 1, t, Pos);
  end;
...


procedure Initialize;
begin
...
  ChartSeriesSetCount(0, 2);
  ChartSeriesSetColor(0, 0, clGreen);
  ChartSeriesSetColor(0, 1, clRed);
  ChartSeriesClear(0, 0);
  ChartSeriesClear(0, 1);
  //ChartSetAxisMinMax(0, -0.05, 1.05, -0.05, 1.05);
  ChartON := False;
  SetRCValue(4,9,'Chart OFF');
```

...