Lab 6

Closed loop vision control

Author:

Jonas Lussi, Tarik Rifai, Franziska Ullrich, Naveen Shamsudhin, and Prof. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Date: 2018

Version: 1.3

Summary: The purpose of these two week's lab is to implement position control using a closed loop controller. We will implement a PID controller and study the effect of proportional and integral control actions on a system. To close the control loop you will use position feedback from your previously written vision-based tracker. Finally, we will use your controller to move a microrobot to a predefined point in space. This week, you will learn how to:

- program your own PID controller
- use a vision-based tracker for feedback
- to do motion control of a magnetic microrobot

6.1 Background

In the next two weeks you will write your own PID controller and use it in a practical microrobotic application, similar to the applications we use in our lab. You will learn how to communicate with stepper motors that move a 2D stage. You get position feedback for your manipulator via a tracker that was implemented in the last lab.

The general setup is depicted in Figure 6.1. A spherical permanent magnetic is glued onto the moving 2D stage. Another small magnetic object (for these purposes we call it "microrobot") moves in a container that is filled with silicone oil. The silicone oil has a higher viscosity than water and has damping characteristics. Thus, it stabilizes the motion of the microrobot. When the stage is actuated and the magnetic sphere is moved on a path, the microrobot will follow the sphere and can thus be manipulated. The motion of the microrobot is observed by a camera and tracked using computer vision. The tracked position acts as a feedback for the closed loop control.

6.2 Motorized Linear Positioning Stage

We actuate the microrobot by controlling a linear 2D motorised positioning stage consisting of two 1D stages (8MT167-25BS, Standa Ltd.), shown in Figure 6.2. Each stage uses a ball bearing lead screw that is powered by a stepper motor (ST2818M1006, Nanotec Electronic). You can find all specifications for the stage and the motor in Tables 6.1 and 6.2.

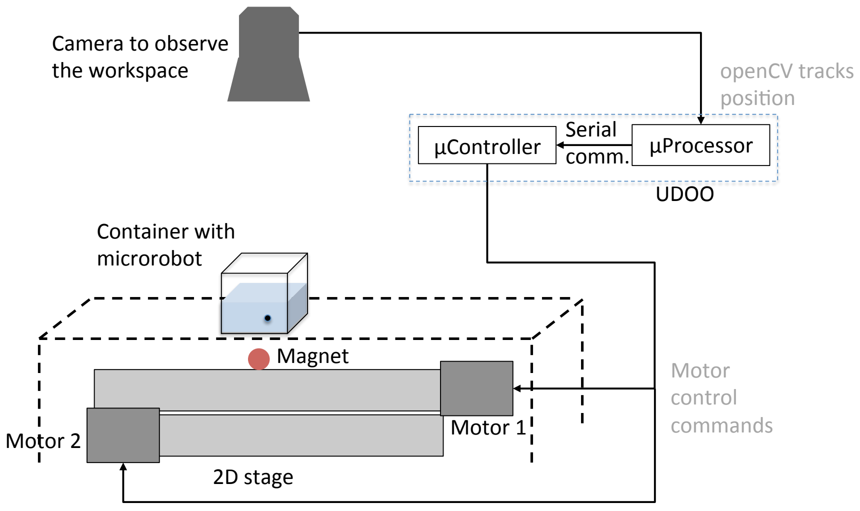

Figure 6.1: Setup for microrobotic application using PID control and visual feedback.

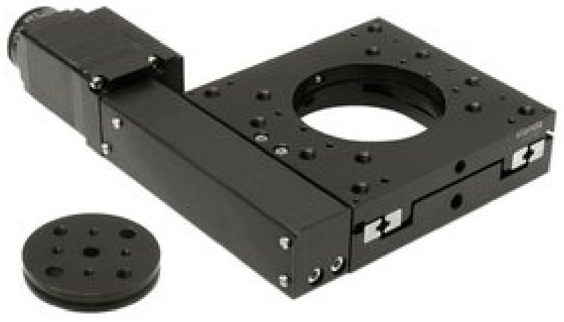

Figure 6.2: Linear motorized positioning stage 8MT167-25BS

6.2.1 Stepper Motor

A stepper motor is a brushless, synchronous electric motor that divides a full rotation into a number of equal steps. Unlike a brushless DC motor which rotates continuously when a fixed DC voltage is applied to it, a step motor rotates in discrete step angles. The motor is controlled by a series of electromagnetic coils. The center shaft has several magnets mounted on it which experience attractive or repulsive forces from the coils when current is switched on or off. This causes the motor to rotate.

The motor rotation has several direct relationships to the applied input pulses. The sequence of applied pulses is related to the direction of motor rotation. The speed of motor rotation is directly related to the input pulse frequency. The length of rotation is directly related to the number of input pulses applied. Using a stepper motor has several advantages:

- the motor shaft rotation angle is proportional to the input pulse and, therefore, an estimate on motor rotation can be used as a feed forward signal
- the motor has full torque at standstill
- the stepper motor can easily be stopped or reversed
- a wide range of rotational speeds can be realised as speed is proportional to input pulse frequency

Table 6.1: Specifications of linear motorized stage 8MT167-25BS1

Travel range	25 mm
Resolution full step	5 µm
Resolution $\frac{1}{8}$ step	0.625 µm
Ball screw pitch	1 mm
Repeatability	1.5 µm
Maximum speed	25 mm/s
Maximum load, horizontal	30 kg
Maximum load, vertical	7 kg
Weight	0.8 kg

Table 6.2: Specifications of stepper motor ST2818M1006

Rotor Inertia	12 gcm^2
Resistance per winding	3.4 Ω
Current per winding	0.95 A/winding
Holding torque	10.61 Ncm
Step angle	1.8° ± 5%
Weight	0.176 kg

You can find more information about stepper motors under http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf.

6.3 Closed Loop System

In the previous week you have written a visual-based tracker that returns the current position of your object. This week we will use position feedback from the tracker to run the 2D stage in closed loop. The output of the controller nr_{steps} is the number of steps that you feed into the stepper motor. The camera measures the position pos_{real} of the microrobot and gives $pos_{measured}$ as the feedback. Figure 6.3 shows the block diagram for the closed loop system.

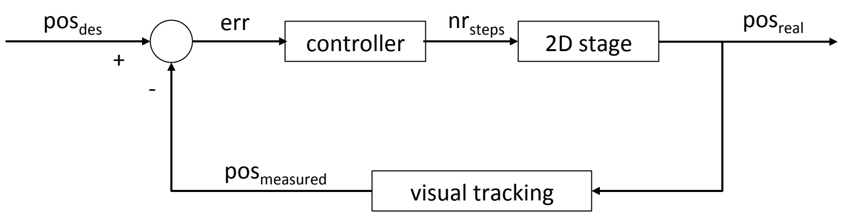

Figure 6.3: Closed loop system with position feedback from the camera.

6.4 PID Controller

The transfer function for an analog PID controller is:

$$T(s) = K_P + \frac{1}{s}K_I + sK_D$$

where K_P, K_I and K_D are the proportional, integral and derivative gains respectively.

Figure 6.4 shows the diagram for a PID controller with anti-windup. In this lab you are not required to implement the anti-windup loop, but you can add it if you want to. We have found good control parameters without the anti-windup.

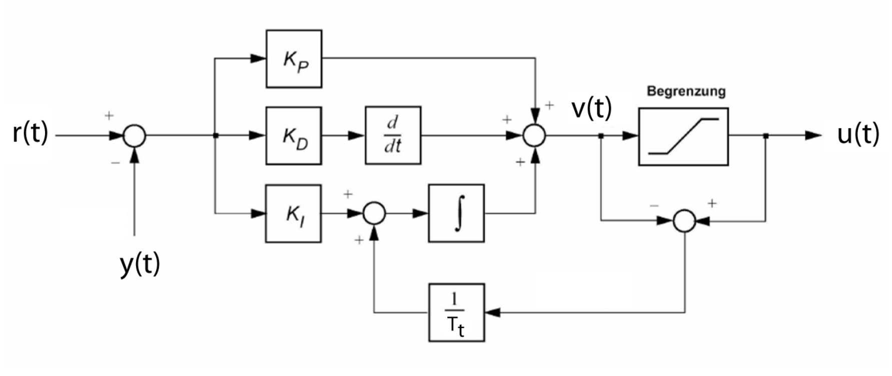

Figure 6.4: PID controller with windup protection [2].

6.4.1 Discrete Implementation

At any time t_n, the output of the controller is given by:

$$v(t_n) = P(t_n) + I(t_n) + D(t_n)$$

where

$$\begin{aligned} P(t_n) &= K_P \cdot e_n \\ I(t_n) &= I(t_{n-1}) + K_I \cdot \frac{e_n + e_{n-1}}{2} \cdot T_s \\ D(t_n) &= K_D \cdot \frac{e_n - e_{n-1}}{T_s} \end{aligned}$$

Note: there are other possibilities to implement the PID controller. For example, a common choice for $I(t_n)$ is also

$$I(t_n) = I(t_{n-1}) + K_I \cdot e_n \cdot T_s \quad \text{or} \quad I(t_n) = I(t_{n-1}) + K_I \cdot e_{n-1} \cdot T_s.$$

The algorithm for PID control thus becomes:

Initialization

1. Set the values of K_P, K_I and K_D.
2. Set the initial values of I_{n-1}, the sample time T_s and e_{n-1}.

Controller algorithm

1. Input the error e_n.
2. Calculate $P(t_n) + I(t_n) + D(t_n)$ using the above equations.
3. Calculate the temporary output $v(t_n)$ using the above equation.
4. Check for saturation and calculate the controller output $u(t_n) = sat(v(t_n))$.
5. Output $u(t_n)$ (used as input to your motor)

Update controller variables

1. Update the value of the error by setting e_{n-1} equal to e_n.
2. Update the value of the position.
3. Update the value of the integral by setting I_{n-1} equal to I_n
4. Wait for the sampling interval to elapse.
5. Repeat the loop.

6.5 Step Response

Depending on the damping in a system and the controller gain that is used, the step response could be oscillatory (under-damped), critically damped or over-damped.

Figure 6.5 shows the step response for different gain values of $K_p > K_{cr}$, $K_p = K_{cr}$ and $K_p < K_{cr}$.

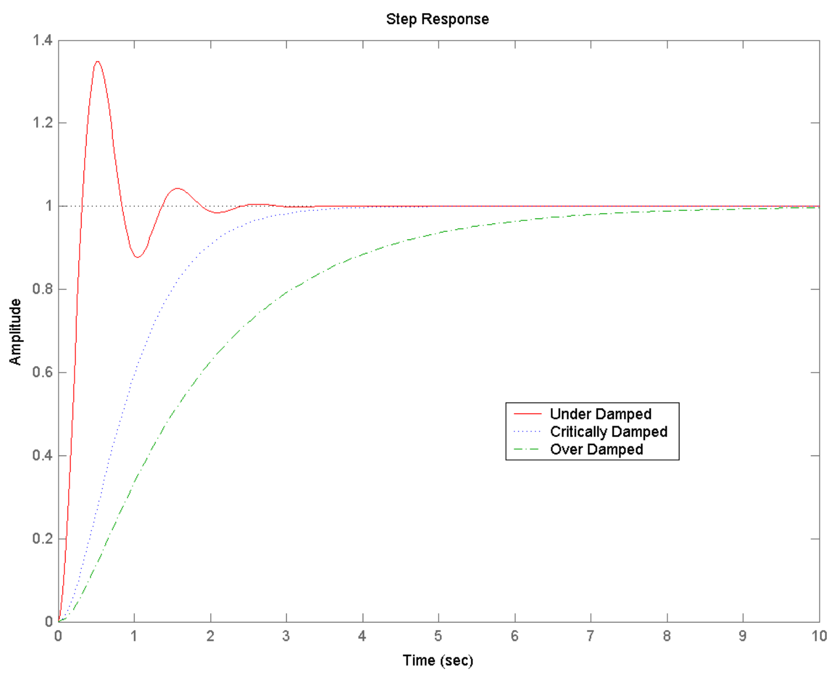

Figure 6.5: response for under damped, critically damped and over damped cases.

6.6 Prelab Procedure

For this lab you will work in both Udoo and Arduino again. The file *lab06.ino* contains the Arduino code and the file *lab06.c* contains the functions written in C code. For the Prelab you will only work on the Arduino code by following these steps:

1. Calculate how many steps per revolution your motors have. **(PreLab Q2)**

2. Open the file *lab06.ino* in the ArduinoIDE environment. Then you need to include libraries, create an `Adafruit_MotorShield` object and create a stepper motor object for each of your motors. Hint: check the documentation for the class `Adafruit_MotorShield` on pages 35 - 37.

3. The linear stages must not move too far and therefore have a limit switch on each side. These switches are connected to the pinouts. The switches are initialized for you. You can find a sketch of the pins and direction of the motors in your folder.

4. Initialise your step size as 20 (as an int).

5. Write a function `void setup()` that starts serial communication and defines the communication timeout. Start the Adafruit Motor Shield and set the maximum speed of the stepper motors to 500. Set your input pins for the limit switches of the linear stages. Hint: Use a baud rate of 115200 /s and timeout of 10 miliseconds.

6. Write a function `int move_steps(int steps,int dir,int motor)` that moves `motor` into direction `dir` with `steps`. First check the motor and its direction and set the limiting switch accordingly. Limit the total number of steps to 999 and move the motor if the set limiting switch is not reached. The function should return the ASCII symbol for '1' if switch 1 is reached, '2' if switch 2 is reached and so on. Return ASCII for '0' if no switch is reached. You will find the ASCII symbols in Table 6.3.

7. Write a function `void loop()` that loops consecutively and allows your program to change and respond. First check if there is a command given to the serial port, this has been implemented for you. A good command contains 5 bytes, where the

   - first byte is the motor number: 1 or 2
   - second byte is the direction: 1 or 2
   - third to fifth bytes are the numbers of steps: 000 - 999

   Determine which motor and direction the command requires and set your parameters accordingly. Then find your number of required steps and save them into your parameter. (Make sure to convert from chars to integers first.) If everything is fine, you can move the motor and return a variable `check` that is sent back to the microprocessor:

   - '0' if motor moved
   - '1' if switch 1 is pressed
   - '2' if switch 2 is pressed
   - '3' if switch 3 is pressed
   - '4' if switch 4 is pressed
   - '5' if the command is bad

8. Hand in all the code (lab06.ino file) you wrote in Tasks 2 to 7. **(PreLab Q2)**

NOTE: Prelab assignments (Prelab Q1 and Prelab Q2) must be done before reporting to the lab and must be turned in to the lab instructor at the beginning of your lab session.

Table 6.3: ASCII symbols

Decimal	ASCII
0	48
1	49
2	50
3	51
4	52

6.7 Lab Procedure

6.7.1 Open Loop Motion

For open loop motion of the 2D stage, we need to setup communication between the UDOO and the motors.

1. In your main function `lab06_test`, remember to first initialise the serial port with `serialport_init()` that has been implemented for you. At the end close the serial port with `serialport_close()`. **(Postlab Q1)**

2. Write a function `constructCommand()` in `pid.c` that creates a command to be sent via the serial port to the microcontroller. Remember that your command is 5 bytes long and that your integers need to be converted to characters first (table 6.3). If successful, the function should return '0'. **(Postlab Q2)**

3. Write a function `moveMotor()` in `pid.c` that allows a motor to move a specified distance in mm. The maximum distance that can be moved is 5 mm. If successful, the function should return '0'. Hint: You can write a command to the serial port with the function `serialport_write()` that has been implemented for you. **(Postlab Q3)**

4. Before using the camera, it needs to be calibrated. For this, move the motor a specified distance and track the spherical magnet with your tracker. This gives you a traveled distance in pixel. Find a conversion factor in [pixel/mm] and note it down. To get a good value, do this experiment 20 times and take the average. Write your code in `lab06_test` Use the color tracker that you implemented in lab04. **(Postlab Q4)**

5. Implement a function `moveMotorRectangular()` in `pid.c` that moves the stages on a rectangular trajectory with a specified distance for each rectangle side. Give an option to the user to use vision. Do this by tracking the spherical magnet that is moved by the 2D stage along a trajectory and saving the coordinates into a .txt file for later use. Use the color tracker that you implemented in lab04. **(Postlab Q5)**

6. Move the magnet on a square trajectories with side lengths 5 mm, and plot the resulting tracked positions of the magnet. Hint: As you are going to run various different postlab tasks in the same mainfile, we suggest to prompt the user first to select a task to be performed. This will also make it easier to show to each task separately to your assistant. **(Postlab Q6)**

6.7.2 PID Controller

7. Write a PID position controller in the `PID()` function (located in in `pid.c`) that uses position feedback from the color tracker. See the specifications for the function in Section "Lab06: PID Controller". In a first step use the P controller (let D and I equal zero). To get the timestamp of each iteration use the function `gettimeofday()` (use it directly to implement your own `tic()` and `toc()` functions.). Make sure to include <sys/time.h>. **(Postlab Q7)**

8. Run the function `PID()` with a desired goal position of x = 5mm (no movement in y). The stop condition for the loop is when the desired position is reached within a given error tolerance. NOTE: the error tolerance must be given in pixel (e.g. 5 Pixels). **(Postlab Q8)**

9. Tune the K_P value of your P controller to find a "good" step response by plotting the position versus the timestamp. Plot out the step response (timestamp versus position). The easiest way to do that, is to write into a .txt file directly in the `PID()` function (similarly to `moveMotorRectangular()`) **(Postlab Q9)**

10. Now we implement also the I and D part of the PID controller according to the formulae above. Tune the parameters, such that you once get a critically damped and an overdamped response. Plot the two step responses (versus timestamp) in one figure. Also try to get the step response for the underdamped system. You might not achieve to get an underdamped system. **(Postlab Q10)**

11. BONUS: Rewrite your functions, such that it controls the x- and y-position simultaneously. (only counts if you don't have max points already) **(Postlab Q11)**

6.7.3 Closed Loop Control

Now we will move the 2D stage using the PID controller.

12. Write code that uses the PID controller to move the spherical magnet along a square trajectory. Again, do this for a square trajectory with side length 5 mm and plot the resulting tracked positions of the magnet. **(Postlab Q12)**

13. What differences do you find when comparing the open loop control to closed loop control Which is better and why? Please explain. **(Postlab Q13)**

6.7.4 Microrobotic Application

14. Place the container with the microrobot on top of the moving magnet. Be careful not to spill any silicone oil as this might damage equipment.

15. Move the magnet on a square trajectory with side length 5 mm in open loop and in closed loop configurations. The microrobot will follow the large magnet on its path. Use the color tracker to track the microrobot. Plot the trajectory. **(Postlab Q14)**

16. Explain how tracking the spherical magnet differs with tracking the microrobot considering the precision of motion. Why do we place the microrobot in silicone oil with a viscosity higher than water? Please explain. **(Postlab Q15)**

6.8 Postlab and lab report

- Hand in the plots of the trajectories in open loop (5 mm side length) and in closed loop (5 mm side lengths), and two trajectories of the microrobot application (5 mm side length). Compare all these trajectories and comment on the differences.
- Hand in the plots for the tuned Values for K_P in Postlab Q9 and the damped/critically and underdamped (if possible) responses in Postlab Q10.
- Print out all your source code files from Postlab Q1-Q15 to turn in.
- Show your running system to the assistant. **(Postlab Q16)**