Introduction to computer based control systems
Process lab 1

Name:
Program:
Date:
Approved: Sign

UPPSALA UNIVERSITY

DEPARTMENT OF SYSTEM AND CONTROL

1 Introduction

There will be three process labs. The goal of the labs is to construct a robotic line tracker that will be able to keep the vehicle at the center of a gradient track in a satisfactory manner.

In this session you will familiarize yourself with the equipment that will be used for the two upcoming labs. First of all, you will build the vehicle that in the end will be able to stay at the center of the track. The next step is to make the vehicle move and to take readings from its sensors. Finally, you will design a control algorithm enabling the vehicle to follow the track.

1.1 Hardware: LEGO NXT

The hardware that will be used is the LEGO NXT. It consists of a NXT-brick which is the brain of the system. There are several sensors and actuators available. The actuators that will be used in this laboration are the DC motors and the sensor that will be used is the light sensor. On the top of the NXT brick there are 3 motor ports A,B and C to which the motors could be connected. On the bottom of the NXT, there are four sensor ports 1, 2, 3 and 4 to which the sensors can be connected. The NXT brick is switched on by pushing the orange square button on the front; to switch it off push the gray button below the orange button and then the orange button.

1.2 Software: RWTH toolbox

RWTH toolbox will be used to "program" the brick. It works so that all the computations are performed on a PC in Matlab and then control signals and sensor readings are communicated between the NXT and the PC via a USB cable. This makes it easy to program and to visualize the data by using Matlab's graphics functions. It is, in principle, possible to use the bluetooth connection for the communication, but it has a delay of about 60 ms which will have negative effects on the achievable control performance.

2 Building the vehicle

The first task is to build the vehicle for the tracker. The vehicle will be a three-wheeled car with a light sensor at the front. There will be two driving wheels, each one attached to a DC motor, while the third wheel is only used as a support wheel.

Task: Build the vehicle by following the building instructions that are provided in the building manual found in the Lego box. When you are done contact the lab assistant to check your design.

3 Preparation

- Open Matlab and get to the lab directory (it will be specified at the lab session).
- Type **addLabPath** in Matlabs command prompt and hit enter to get access to all required functions and m-files. Note that you should stay in this directory during the whole lab!
- Connect the NXT and PC with the USB cable.
- Turn the NXT on by pushing the orange button.
- Type **h=init();** in Matlabs command prompt and hit enter. If you receive no error messages everything is working correctly and you have managed to establish a connection to the NXT.
- Type **closeConnection(h);** in Matlabs command prompt and hit enter to close the connection to the NXT.

4 Actuators

In a system, an actuator is a device that is able to actuate a control signal. In the vehicle, the actuators will be the two DC motors that can move the system provided input voltages are applied to them. This can be done by calling the function **applyMotorVoltage** (see Appendix). Note that this function takes as input the percentage of the maximum voltage to be applied to the motor.

Task: Use the m-file **actuatorTest.m** as a start, open it by typing **open actuatorTest** in Matlab's command prompt. Make sure that the right and left motor are connected to the corresponding ports. Fill in the missing commands to make an m-file that:

- Establishes a connection to the NXT.
- Applies a voltage of 50 % of the maximum available voltage on both left and right motor.
- Run for 3 seconds.
- Stops, i.e. applies 0 % voltage to both motors.
- Closes the connection to the NXT.

Question: Even with the same voltage applied to both motors, the vehicle is not running straight. Give suggestions to why this might be the case.

From now on, the vehicle will be controlled by applying an offset voltage u_0 to both motors causing the vehicle to go forward at constant speed, and the steering will be performed by applying a differential voltage Δu to the motors. If u_l and u_r is the voltage applied to the left and right motor they will be given

$$\begin{aligned} u_l &= u_0 + \Delta u/2 \\ u_r &= u_0 - \Delta u/2 \end{aligned}$$

The function **applyDeltaVoltage** (see Appendix) should be used to apply these voltages to the motors.

Task: Rewrite **actuatorTest.m** using the function **applyDeltaVoltage** when applying the motor voltages. Try out different values of Δu, u_0 and observe the behavior.

5 Sensor

Some kind of input is required to know where on the track the robot is. The sensor that will be used to measure the distance to the center of the track is a light sensor. It is able to measure the light intensity it is receiving. As you can see, the track consists of a gradient line. Since bright colors reflect more light than dark colors, it will be possible to determine the vehicle position on the track by measuring the light intensity registered by the sensor.

The sensor can (and should) be run in active mode. It will then emit light through a LED (light emitting diode) and make the sensor reading less sensitive to ambient light. By calling **init**, the light sensor will automatically be set in active mode provided that it is connected to sensor port 1.

5.1 Reading the sensor

To read a value from the sensor, use the **GetLight** command (see Appendix).

Task*:** Use ***sensorTest.m as a start (open this by typing **open sensorTest** in Matlabs command prompt) and fill in the missing commands to produce an m-file that does the following:

- Initializes a connection to the NXT
- Reads the light intensity value and prints it in Matlabs command prompt.
- Closes the connection to the NXT.

5.2 Calibrating the sensor

The light sensor measures the received light intensity. What we actually want to know is the distance to the center of the track. This means that a calibration is required to transfrom the light intensity to a distance. When the sensor is within the track, the received light intensity l depends linearly on the distance y from the center of the track, i.e. $l = a_1 y + a_0$. To calibrate the sensor, the parameters a_0 and a_1 have to determined.

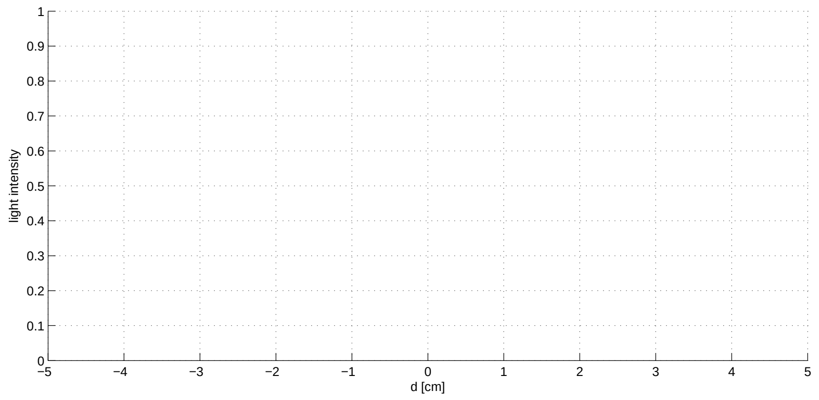
Task: Calibrate the sensor by the following steps:

- Use a ruler and a pencil to mark the points $y = \{-10, -5, 0, 5, 10\}$ mm on the track, ask the lab assistant how this should be done.
- Run the function *[y, l] = calibrateLS(y)*, see Appendix. As input, give a vector of the measurement points in m i.e. $y = 10^{-3}[-10, -5, 0, 5, 10]$. As output you will receive the y vector together with a vector containing the corresponding light intensity values l. It will ask you to put the middle of the sensor at the given points $y = \{-10, -5, 0, 5, 10\}$mm and press enter for each reading. Make sure that the center of the sensor is located at the given position when the measurement is taken.
- Make a plot of y versus l, i.e. **plot(y,l)**.
- Now determine the parameters a_0 and a_1 by fitting a line using the least squares method. Matlabs polyfit function should be used here, type *help polyfit* in Matlabs command prompt for help. Write down these values for later use.
- Plot the measured data points together with the fitted line by calling the *plotCalCurve* (see Appendix) function to make sure your calibration is valid. Show the calibrated curve to the lab assistant.

Question: Does it seem to be a linear relationship between the light intensity and the distance from center of the track?

Question: What will the formula to convert a light intensity value to a distance to the center of track be?

Question: What will happen if the sensor comes too close to the edge of the track, will there still be a linear relationship between l and y? Sketch the light intensity curve in the plot below assuming that the track is 5 cm wide and that the maximum received intensity is 1 when the sensor is reading all white and 0 when reading all black.



6 Control

You now know how to control the motors and read the light sensor. To make a line tracker, the system will be set up so that the left and right motor are given an equal offset voltage u_0 causing the vehicle to move forward at a constant speed and the steering will be performed by applying a differential voltage Δu

to the motors. The voltages applied to the left and right motor u_l and u_r are given by

$$\begin{aligned} u_l &= u_0 + \Delta u/2 \\ u_r &= u_0 - \Delta u/2 \end{aligned}$$

The system thus has one input signal Δu and one output signal y.

A control law is an function (or algorithm) that takes the sensor measurements as input and from these values calculates a control signal to the system that should give the desired behavior.

Task: The m-file *lineTracker.m* can be used as a start for this task. Open it by typing *open lineTracker* in Matlabs command prompt.

Your task is to make a simple control law that takes as input the distance from center of the track y and produces a differential voltage Δu that is applied to the motors. The function **applyDeltaVoltage** should be used to apply the input signal Δu, see Appendix. Use the offset voltage $u_0 = 30$ as input to **applyDelta Voltage**.

The code implementing such a control law should include the following steps:

- Initialize a connection to the NXT.
- Do for 100 iterations (Control loop)
  - Read the light sensor.
  - Convert the read intensity to a distance from the center of the track.
  - Calculate a control signal Δu according to your control law.
  - Apply the calculated control signal to the motors.
- Stop the motors.
- Close the connection to the NXT.

Fill in the missing commands and execute the code to see whether the vehicle is able to stay on track using your control law.

Note: It is likely that there will be some failures along the way of finding a control law that stabilizes the vehicle on the track, causing the vehicle to spin and twist the usb cable. In that case, disconnect the usb cable when the program is finished executing unwind the cable and reconnect it.

When you have a solution that is able to keep the vehicle on track you are done with the lab.

7 Appendix

```
% ***********************************************************************
%
% function applyMotorVoltage(MOTOR_L, u_l, MOTOR_R, u_r)
%
% Input
%   MOTOR_L: Port for left motor (MOTOR_{A,B,C})
%   MOTOR_R: Port for right motor (MOTOR_{A,B,C})
%   u_left : Percentage of maximum voltage to apply to left motor
%   u_right: Percentage of maximum voltage to apply to right motor
%
% Applies u_left*U_MAX and u_right*U_MAX volts to left respectively right
% motor

% ***********************************************************************
%
% function plotCalCurve(y_points, l_points, a0, a1)
%
% INPUT
%   y_points: Vector containing the measurement points in m
%   l_points: Vector whe element i is the light value read at position y(i)
%   a0:       Offset parameter
%   a1:       Slope parameter
%
% Plots the measurement points (y(i), l(i)) i=1..length(y) together with
% the fitted curve y=a1*l + a0.

% ***********************************************************************
% function applyDeltaVoltage(u0, delta_u, MOTOR_L, MOTOR_R)
%
% INPUT:
% u0        : Offset voltage in percent of max voltage
% delta_u   : Differential voltage in percent of max voltage
% MOTOR_L   : Port for left Motor (MOTOR_{A,B,C})
% MOTOR_R   : Port for right Motor (MOTOR_{A,B,C})
%
%
% Applies (u0 + delta_u/2)*U_MAX and (u0 - delta_u/2)*U_MAX volts to
% MOTOR_L respectively MOTOR_R.
%

% ***********************************************************************
% function closeConnection(h)
%
% Input:
%   h: A hande to the NXT for which the connection should be closed.
%
% Stops the motors, turns of the light sensor and closes the connection to
% the NXT associated with the handle h.
%
% ***********************************************************************
% function applyMotorVoltage(MOTOR_L, u_l, MOTOR_R, u_r)
```

```
%
% Input
%   MOTOR_L: Port for left motor (MOTOR_{A,B,C})
%   MOTOR_R: Port for right motor (MOTOR_{A,B,C})
%   u_left : Percentage of maximum voltage to apply to left motor
%   u_right: Percentage of maximum voltage to apply to right motor
%
% Applies u_left*U_MAX and u_right*U_MAX volts to left respectively right
% motor

% *********************************************************************
%
% function [y, l] = calibrateLS(y)
%
% INPUT:
%   y: A vector specifying the callibration points in m
%
% OUTPUT:
%   y: A vector specifying the callibration points in m
%   l: A vector containing the light intensity at the corresponding y
%   points.
%
% Asks the user to place the sensor at d(i) mm from the center of the
% track, i=1,..lenght(d) and press enter at each point to take a
% corresponding light intensity reading y(i).
%

% *********************************************************************
% function h = init()
%
% OUTPUT
%   h: A handle to the NXT brick connected to the usb cable
%
% Closes all current connections to the NXT and opens a new connection.
% Make sure that the NXT is powered on and connected to the USB cable when
% calling this function.

% *********************************************************************
%   Reads the current value of the NXT light sensor
%
%   Syntax
%     light = GetLight(port)
%
%     light = GetLight(port, handle)
%
%   Description
%     light = GetLight(port) returns the current light value light of the
%     NXT light sensor. The measurement value light represents the
%     normalized (default) sound value (0..1023 / 10 Bit). The normalized
%     value mode is set per default by the function OpenLight. The given
%     port number specifies the connection port. The value port can be
%     addressed by the symbolic constants SENSOR_1 , SENSOR_2, SENSOR_3 and
%     SENSOR_4 analog to the labeling on the NXT Brick.
%
%     The last optional argument can be a valid NXT handle. If none is
%     specified, the default handle will be used (call COM_SetDefaultNXT to
%     set one).
%
%     For more complex settings the function NXT_GetInputValues can be
%     used.
```