EL2450
Hybrid and Embedded Control Systems
Homework 3

[To be handed in **February 27**]

1 Introduction

The goal of this homework is to control a differential-drive robot to make it follow a pre-defined behavior in the workspace. This homework is designed for a **group of 4 students**, where the workload should be evenly distributed.

The homework consists of two parts; a report (on Part 4) where 44 points are available, and a lab session (Part 5) which is pass/fail only. To book a lab-session you must have a pass on the report, i.e. achieved at least 36 points. To have a chance to resubmit the report after the first hand-in you must achieve at least 18 points on the initial hand-in.

Make sure that all group members have read this manual entirely and that you have successfully completed all the tasks before Part 5. Please follow the procedure described on the page *Instructions for Homework* on Canvas to book the lab session. Before you start working on the tasks, we recommend that you go through this manual one time from top to bottom.

In Part 3, you are given some information about the hardware and software setup that you are going to use for the homework. In Part 4.1, you are asked to analyse a simple mathematical model of the robot kinematics. In Part 4.2, you are asked to abstract the motion of the robot as a finite Transition System $\mathcal{T}$ and derive a high-level plan that satisfies a given specification.

In Part 4.3, you are asked to design and implement a hybrid controller that enables the robot to follow the trajectory that you have planned in Part 4.2. First, you are asked to design two simple controllers: one to rotate the robot to a certain orientation and the other to navigate the robot along a straight line. Then, you are asked to combine these two controllers into a hybrid controller that enables the robot to follow an assigned trajectory in a corridor-like environment. A real world application of this controller could be, for example, a robot that brings medicine to patients in a hospital. The robot cannot drive around like it wants to but has to follow the corridor network of the hospital. Therefore, the robot first rotates until it faces to the point in the corridor it

wants to reach and then follows a straight line to this point. If this point is, for example, the room of a patient, the robot stops. Otherwise it starts the procedure again to go to another point in the corridor environment until it reaches its actual goal.

Finally, in Part 5, you are asked to come to the *Smart Mobility Lab* and test your controller on a real differential-drive robot.

2 Report

The report for this homework should follow the template provided at the course webpage on canvas. The report should contain answers to the first 19 tasks, and should be submitted on Canvas within the deadline specified on the `Homework` page. The remaining tasks (Part 5) are going to be discussed directly during the lab session, and require no additional submission. Each group should submit a single report. Do not repeat the instructions nor the questions, but motivate well and support your answers, with theoretical analysis (for Tasks 1, 2, 3, 4, 5, 7, 10, 12, 16), and with simulation or experimental results (for Tasks 6, 8, 9, 11, 13, 14, 15, 17, 18, 19, 20, 21, 22).

Please, do your best to stay within a page limit of 15. If you exceed this limit too blatantly, we may ask you to resubmit a shortened version.

This homework is a complex and heterogeneous challenge, and the TAs heavily update it every year to make your experience as painless as possible, but at the same time, instructive and challenging. Therefore, you are kindly encouraged to add, at the end of your report, any comments and suggestions about this homework, which will be greatly helpful to improve it for the coming years.

2.1 Responsible TAs for questions

The responsible TAs are:

1. **Victor Molnö, vmolno@kth.se**
2. **Adrian Wiltz, wiltz@kth.se**

2.2 Upload Instructions

Each group uploads a single file on canvas, with the deadline specified on Canvas. This file needs to be one zip file with `name name1-name2-name3-name4-HW3.zip`, where `name1,...,name4` are the last names of the authors. The zip file should contain 4 files: one pdf file named `name1-name2-name3-name4-HW3.pdf` and three controller files `OwnVariables.c`, `Controller.c` and `RenewControllerState.c`.

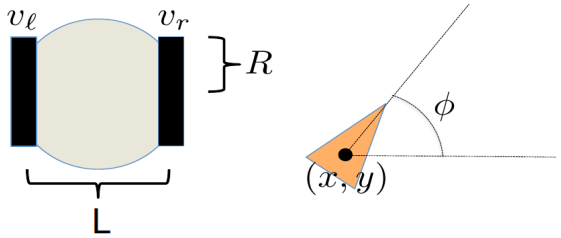

Figure 1: Differential-drive robot

3 Setup Description

3.1 Setup in the Lab

The robot used for this homework is a two-wheeled differential-drive robot of the brand NEXUS (see Figure 1). The NEXUS is equipped with an Arduino board to control the motors and access the onboard sensors, and with a wireless transreceiver to exchange information.

As shown in Figure 2, the position and orientation of each of the robots is measured with the QUALISYS motion capturing system in real-time (also known as MOCAP), which has the structure (`timestamp, x, y, theta`). Note that `timestamp` is measured in milliseconds; `x, y` are in meters; `theta` is in degrees in the range $(-180°, 180°]$. The information about the position and orientation of the robot is provided over the local network by a computer `M`, which is connected to the MOCAP system. This information is then sent to another computer `H` (the host) which, in turn, transmits it wirelessly to the robot. For debugging and documentation purposes, it is possible to send information back to computer `H`, which displays these messages.

A *graphical user interface* (GUI) like the one shown in Figure 3 is provided on computer `H`. The GUI allows you to

- manually control the robot,
- activate the automatic controller,
- record the robot's position data, and
- specify the start and goal states for the robot

on the fly.

The logging function is useful when analyzing the robot's trajectory and writing the final report. Specifically, the Arduino program will receive different messages wirelessly from the computer `H` (controlled by you through the GUI). Arduino will parse these messages and save the information in different variables. For instance, it receives

- `MANUAL_CONTROL` messages that contain the manual control inputs you have specified;

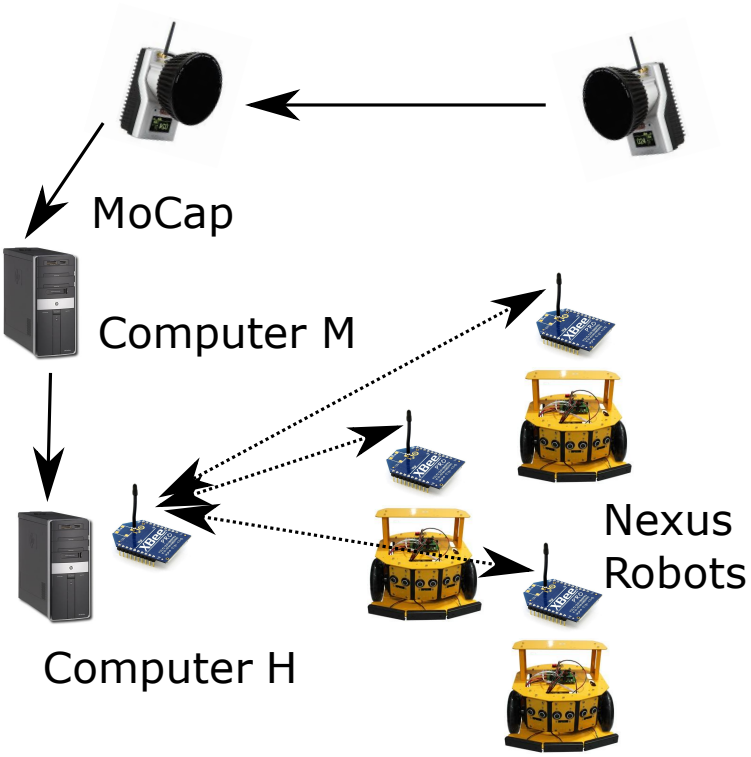

Figure 2: The hardware setup

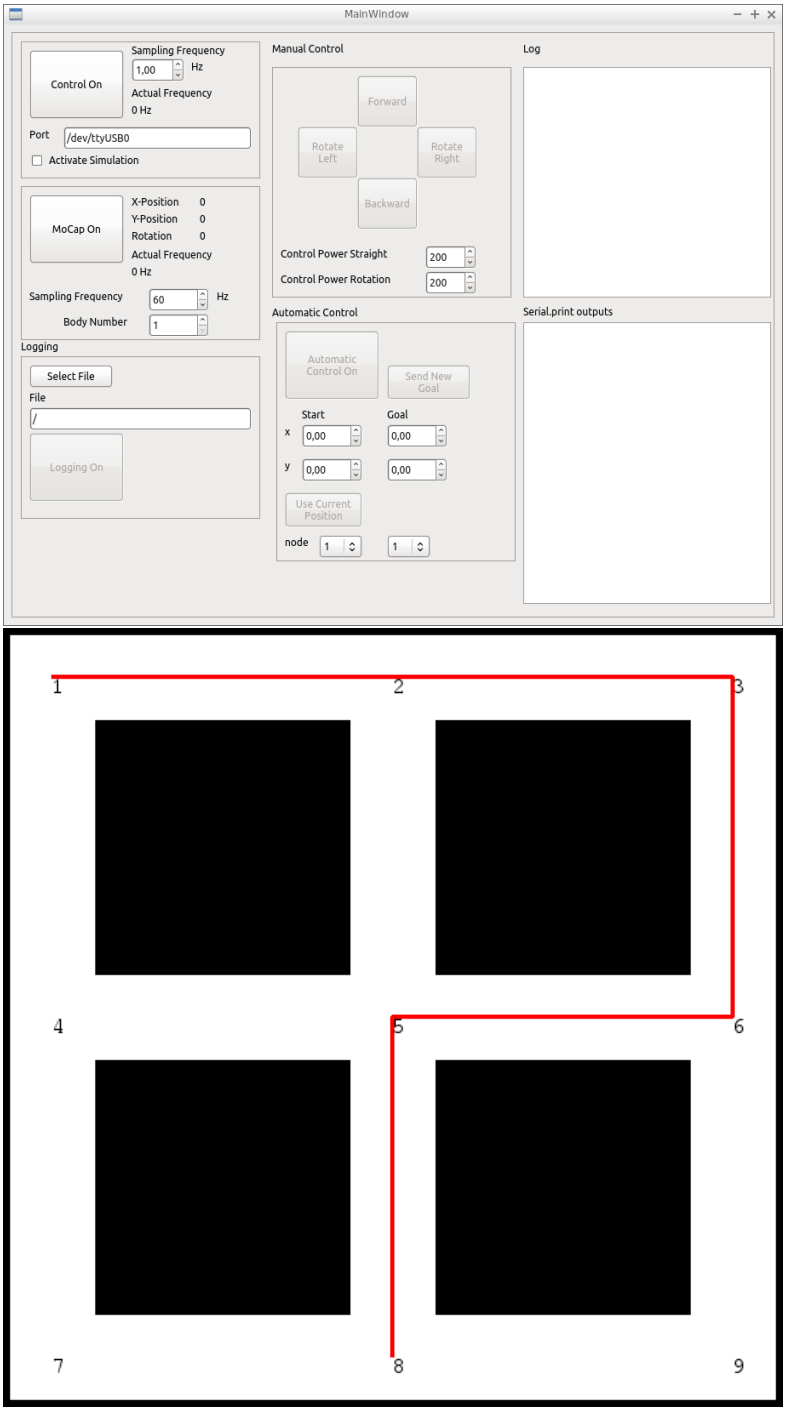

Figure 3: Top: the GUI. Bottom: The virtual corridor environment.

- `POSE_DATA` messages that contain the current state of the robot, i.e., (x, y, θ);
- `START_GOAL` messages that contain the start and goal locations you specify, i.e., (x_0, y_0, x_g, y_g).

For the ease of wireless encoding and transmission, the measurements of the robot's position by the MoCap system are converted to centimeters before they are transmitted to the robot. Therefore, the GUI sees the robots state in meters and degrees, while the robot itself sees it in centimeters and degrees. Be careful to convert the measurements appropriately in the controller.

Most of the code that you need to run on the Arduino is already provided for you. You are only asked to fill in three gaps, called `OwnVariables`, `Controller`, and `RenewControllerState`. When you come to the lab, you should only write your code in this gaps, and you leave everything else in the provided code as it is.

3.2 Setup for the simulations

To make it easy to test your controller, we have provided an online simulator of the real robot. To use the simulator, you only need to navigate to a given address from a web browser. We will upload several web addresses on Canvas for that. You can use all of them. However, if you encounter lagging or crashes, please change to another address (this is likely to happen due to the potential overload if it is used by a lot of people simultaneously). **Please use the Google Chrome browser** (Other browsers may be missing some functionalities that are necessary to run the simulator.) There is also a video tutorial that illustrates how to use the simulator. A link to the tutorial will be given on Canvas. The online simulator has the same GUI that you will find on the computer H in the lab; therefore, if you learn to send commands to the simulated robot, you will be able to interact in the same way with the real robot in the lab.

The simulator understands the same code as the real robot; therefore, after you have tested your controller on the simulator, you can simply copy-paste it on the real robot during the lab session. However, while in the real robot you have to fill in the three gaps `OwnVariables`, `Controller`, and `RenewControllerState`, in the online simulator, you have to provide three files `OwnVariables.c`, `Controller.c`, and `RenewControllerState.c`. Each of this files must contain the code that you would place in the corresponding gap.

You will be able to reach the simulator online, and you do not need to download any specific program to run it. The web address will be communicated on Canvas.

3.3 Logging and Plotting

Both the real robot GUI and the online simulator provide a functionality (`Logging On` to save a file with a record of the robot position and control signals. Although MATLab is not used directly in this homework, you can load these files in MATLab to make plots and document the performances of your controller, when you are asked to do so in a Task. In fact, when a Task asks you to evaluate the performance of a controller, you

are likely to lose some points if you do not include at least one plot. (However, make sure that you do not exceed the page limit.) The file format is `csv`, and it is easily readable. The MATLAB function `dlmread()` might be helpful to generate the plots. Newer MATLAB versions have also a graphical tool for reading data out of files.

4 Homework Tasks to be Submitted in the Report

4.1 System Modeling

The robot is shown in Figure 1 and can be modelled as follows:

$$\begin{aligned} \dot{x} &= \frac{R}{2}(u_l + u_r)\cos\theta, \\ \dot{y} &= \frac{R}{2}(u_l + u_r)\sin\theta, \\ \dot{\theta} &= \frac{R}{L}(u_r - u_l), \end{aligned} \tag{1}$$

where $[x,\ y,\ \theta]$ is the robot's state, including its 2-D location and orientation with respect to an inertial frame of reference. An accurate estimate of R and L for the real robot is given in the code that we provide for you as the variables `R_true` and `L_true`. Note that x and y are measured in meters, while θ is in degrees in the range $(-180°,\ 180°]$. u_l and u_r are the angular velocities $(1°/s)$ of the left and right wheel, respectively; R is the wheel radius (m) and L is the distance between the two wheels (m).

Alternatively, let

$$\begin{aligned} v &= \frac{u_r + u_l}{2}, \\ \omega &= u_r - u_l, \end{aligned} \tag{2}$$

be the input to *translate* the robot and to *rotate* it, respectively. As a result, (1) can be rewritten as

$$\begin{aligned} \dot{x} &= R\,v\cos\theta, \\ \dot{y} &= R\,v\sin\theta, \\ \dot{\theta} &= \frac{R}{L}\,\omega. \end{aligned} \tag{3}$$

Task 1 (2p)**.** *From the above equation, it seems more convenient to calculate the velocities v and ω for the control of the vehicle. However, the actual velocities that will be implemented by the vehicle are u_r and u_l. Hence, how can you compute u_r and u_l, if v and ω are given?*

Before we move on, it would be good to play some time with the simulator by controlling the robot manually. In fact, controlling the robot manually will give you an intuitive understanding of the robot kinematics. Please refer to the video tutorial of the simulator to learn how to control the robot manually. Since you don't need a controller for this, you can use the empty files contained in the `EmptySolution` folder to run the simulator.

4.2 Motion Planning with Transition Systems

In this section we are going to derive a trajectory for the robot to follow a predefined specification.

The first step is to abstract the continuous motion of the robot to a finite transition system $\mathcal{T}$. A Transition System is a tuple

$$\mathcal{T} = (S, S_0, \Sigma, \rightarrow, AP, \mathcal{L}) \tag{4}$$

where S is set of states, $S_0 \subseteq S$ is a set of initial states, Σ is a set of actions, $\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation, AP is a finite set of atomic propositions, which are statements over the problem variables and parameters that can be either `True` or `False`, and $\mathcal{L} : S \rightarrow 2^{AP}$ is a labeling function, that assigns to each state $s \in \mathcal{S}$, the subset of atomic propositions AP that are `True` in that state. Given a path $p = s_1, s_2, \ldots$ of $\mathcal{T}$, with $s_1, s_2, \cdots \in \mathcal{S}$, its Trace is defined as $\text{Trace}(p) = \mathcal{L}(s_1)\mathcal{L}(s_2)\ldots$, i.e. the atomic propositions that are true along the path. In this homework, we will discretize the workspace of the vehicle and approximate it as a transition system.

First, we need to provide a discretization of the workspace into regions that will resemble the states of our $\mathcal{T}$. The discretization that we consider is $R = \{R_1, \ldots, R_K\}$, i.e., K equally sized rectangular regions, as illustrated in Figure 4. The workspace that the robot operates is a 2D rectangle, centered at $(0, 0)$, with dimensions $3 \times 3\text{m}^2$.

Moreover, we consider a set of properties of interest that hold in some of the regions of the workspace. For simplicity, we assume that these properties are "blue", "red", "green" as well as "obstacle" for potential obstacles in the workspace. The obstacles are small spheres of radius 0.05m and are centered at the positions $(-0.75, 0.75)$, $(0.25, 1.25)$, $(0.75, 0.75)$, $(0.25, -0.25)$, $(0.75, -0.75)$, $(-0.75, -0.75)$, $(-0.75, -0.25)$, $(1.25, 1.25)$ and $(-1.25, -0.25)$. Moreover, we consider that:

- "red" holds at the positions $(-0.75, 0.7)$, $(0.2, 0.7)$, $(1.25, 1.25)$, $(1.2, 0.8)$, $(0.8, -0.8)$, $(-0.9, -0.8)$, $(-1.25, -1.25)$,
- "blue" holds at the positions $(-0.75, 1.4)$, $(0.9, 0.9)$, $(0.3, 0.2)$, $(0.25, -0.75)$, $(1.2, -1.4)$,
- "green" holds at the positions $(-1.23, 1.25)$, $(1.25, 1.25)$, $(-0.9, 0.2)$, $(-1.2, -0.7)$, $(0.6, -1.2)$.

We assume that the starting position of the robot is always at $(-1.25, 1.25)$. (The position of the robot is computed at its center of mass.)

Task 2 (8p)**.** *Given the aforementioned specifications, and the fact that the largest dimension of the robot is approximately* 38 *centimeters, propose a Transition System of the form* $\mathcal{T} = (S, S_0, \Sigma, \rightarrow, AP, \mathcal{L})$*. Specify also the number* K *of regions as well as its dimensions. The numbering of the regions should be done as in Figure 4. Also, take into account the fact that during a transition between two regions, the robot should always remain in these two regions. Finally, provide a graphical illustration of the discretized*

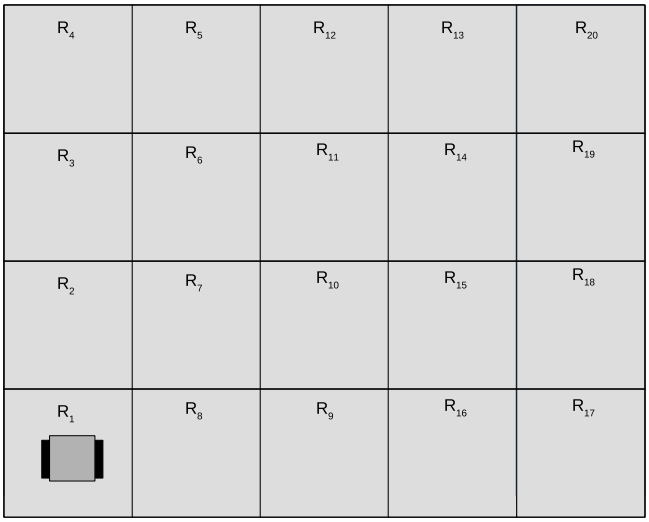

Figure 4: Workspace discretization in rectangular regions

workspace (as in Figure 4) along with the atomic propositions Π *that are true in each state.*

Also, the following definition might be useful:

Consider region R_i *and its center* c_i*, for some* $i \in \{1, \dots, K\}$*. The set of neighboring regions of* R_i *is* $\mathcal{N}_i = \{j \in \{1, \dots, K\} \backslash \{i\},\ s.t.\ \|c_i - c_j\| = d_R\}$*, where* d_R *is the dimension of each region.*

In order for the transition system to make sense, we need to guarantee that the transitions $\rightarrow$ from one region to another can be performed by the action set Σ. Therefore, we need to design appropriate control inputs that take the robot from one region to the other, which is the main concern ot the next section.

For now, assume that the transition system $\mathcal{T}$ is valid, i.e., there exists continuous controllers that guarantee the transitions $\rightarrow$ you defined in Task 2.

Task 3 (4p)**.** *Consider the desired robot behavior:*

- *The robot needs to visit a "red" region infinitely often.*
- *After visiting a "red" region, the robot must visit a "blue" region (at the next state).*
- *The robot should never go to a "obstacle" regions.*

Find a path, as an infinite sequence of states of the Transition System, that satisfies the aforementioned behavior. The "infinity" in the sequence will be written as a finite prefix followed by a finite suffix, which will be run infinite times.

Table 1: Already implemented variables in C/C++ code

Symbol	Variable Name	Type	Description	Changable
x	x	int	x position of the robot	NO
y	y	int	y position of the robot	NO
θ	theta	int	orientation of the robot	NO
x_0	x0	int	initial x position	NO
y_0	y0	int	initial y position	NO
x_g	xg	int	goal x position	NO
y_g	yg	int	goal y position	NO
u_l	left	int	left control value	YES
u_r	right	int	right control value	YES

4.3 Controller Design and Implementation

In this part, you are asked to design and implement a hybrid controller to navigate the differential-drive robot from any initial position in one region to another region, according to the transition relation you defined in Task 2. A safe way to do that, in the sense that the robot does not cross the boundaries with other regions (as specified in Task 2), is to design the controller such that it drives the robot from a region to the center of the goal region.

The hybrid controller has two discrete states:

- the *rotation controller*, which rotates the robot to a certain orientation while staying close to its initial position;
- the *line following controller*, which navigates the robot from one point to another, while staying close to the straight line connecting those two points.

To implement the controller, you are asked to program in C/C++, but in case you do not know C/C++ or need to refresh your memory, you can look into `CBasics.pdf` provided on the course web. `CBasics.pdf` provides you with all the information that you need to implement the controller in C/C++.

You can implement the controller by filling in the three files `OwnVariables`, `Controller`, and `RenewControllerState`. These files are used to test the controller in the online simulator, but you can copy-paste their content in the real robot controller as well.

It is highly recommended that you implement and extensively test each part of your controller before designing the next part. If you design all the parts first, then it will be harder to implement all the features from scratch.

Inside the file called `OwnVariables.c`, you have to declare all the extra variables you want to use (in addition to those that are already declared for you.) The variables already defined inside the code and are listed in Table 1.

Except for the variables *left* and *right*, the other predefined variables in Table 1 are constants: you are not supposed to change their values at any time.

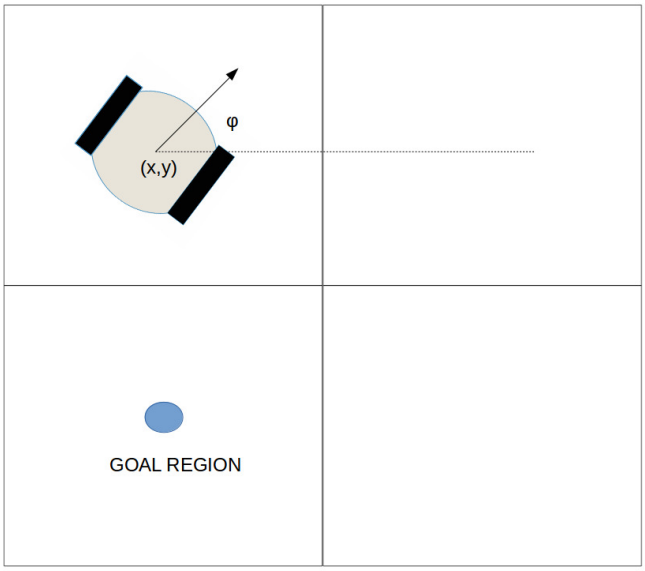

Figure 5: The mobile Robot and the goal region for a potential transition.

The second file is `Controller.c`. Here you implement everything which belongs to the controller itself, so no declaration of variables. This means that you calculate v and ω here to determine u_l and u_r and later decide in which controller state the controller has to be.

Later, you will use the last file `RenewControllerState.c` to renew the controller state, when you connect the two states of the hybrid controller.

In the following, we will implement a hybrid controller as follows. First, we will first design a rotation control that aligns the robot with the line connecting the robot to the goal (i.e., fixes the robot orientation). Then, we will design a *go-to-goal* controller, that actually drives the robot to the goal region of the transition.

Task 4 (1p)**.** *Explain how this choice is related to the "safe" property that during one transition, the robot must always remain in the two regions of the transition (the one it starts from and the one it wants to go to). Would it be better to simultaneously control both the relative orientation and distance of the robot to the goal region? Motivate your answer. (Look at Figure 5).*

4.3.1 Discrete-Time Rotation Control

The objective of the rotation controller consists of two parts:

1. Controlling the rotation velocity ω, change the robot's orientation θ such that it is close to a desired angle θ^R;

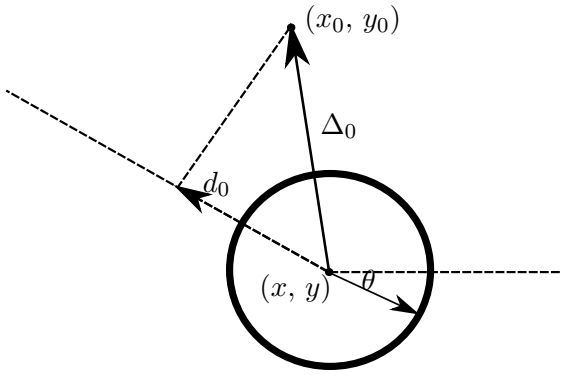

Figure 6: d_0 as the inner product between Δ_0 and $v_c[k]$.

2. Controlling the translation velocity v, maintain the robot's location $[x,\, y]$ close to its initial position $[x_0,\, y_0]$ when it starts rotating, i.e., maintain its location constant.

Note that v and ω are defined by (2). Regarding Part (I), design a proportional controller

$$\omega[k] = K_\Psi(\theta^R - \theta[k]); \tag{5}$$

such that $\theta[k]$ approaches θ^R asymptotically.

Task 5 (2p)**.** *What is the maximal value of K_Ψ to achieve the goal? Prove your statement formally. How do you choose K_Ψ? Please motivate your choice.*

Task 6 (2p)**.** *Implement the controller and show the performance by simulation results when you set $v = 0$. Is it possible to maintain $\theta[k]$ at θ^R exactly? Why?*

Regarding Part (II), design a proportional controller

$$v[k] = K_\omega\, d_0[k]; \tag{6}$$

where $d_0[k] = (v_c[k]^T)\,(\Delta_0[k])$; $v_c[k] = [\cos(\theta[k]), \sin(\theta[k])]^T$ and

$$\Delta_0[k] = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \begin{bmatrix} x[k] \\ y[k] \end{bmatrix}.$$

As shown in Figure 6, $d_0[k]$ is the inner product between the vector from $[x_0,\ y_0]$ to $[x[k],\ y[k]]$ and the vector $v_c[k]$. In order to keep the robot close to $[x_0,\ y_0]$, we want $d_0[k]$ to approach 0 asymptotically.

Task 7 (2p)**.** *Assume that $\theta[k+1] = \theta[k]$, i.e., θ does not change. Derive the equation to show how $d_0[k]$ changes with time. What is the maximal value of K_ω for $d_0[k]$ to converge to 0? Prove your statement formally. How do you choose K_ω? Please motivate.*

Task 8 (2p)**.** *Implement the controller and show the performance by simulation results when you set $\omega = 0$. Is it possible to maintain $[x[k],\ y[k]]$ at $[x_0,\ y_0]$ exactly? Why? (Hint: Bear in mind that $\omega = 0$ and think about what you want to test/show here!)*

Assume that you have obtained the two controllers from above. Now we want to combine them in order to achieve the rotation control objectives mentioned earlier. Note that ω and v are controlled separately by (5) and (6).

Task 9 (2p)**.** *Evaluate the control performance by simulation when both controllers are enabled. Plot the errors $\theta^R - \theta[k]$ and $d_0[k]$. Do they evolve differently compared with when only one controller is enabled? Why?*

4.3.2 Go-To-Goal Controller

When the robot is at the desired orientation θ_g, it should head to the desired goal point (x_g, y_g), which is the center of the transition goal region. In order for the robot not to decline from the line connecting its center to (x_g, y_g), a reasonable choice is to design a line-following algorithm that will aim to keep the robot always on the line connecting its center of mass with (x_g, y_g).

Now we want to design a line following controller which also consists of two parts:

I controlling the translational velocity v, drive the robot from a start location $[x_0,\ y_0]$ to any goal location $[x_g,\ y_g]$;

II controlling the rotational velocity ω, keep the robot close to the straight line connecting $[x_0,\ y_0]$ and $[x_g,\ y_g]$.

Regarding part (I), design a proportional controller

$$v[k] = K_\omega\, d_g[k]; \tag{7}$$

where $d_g[k] = (v_g^T)\,(\Delta_g[k])$; $v_g = [\cos(\theta_g),\ \sin(\theta_g)]^T$,
$\theta_g = \angle([x_0,\ y_0]^T, [x_g,\ y_g]^T)$ and

$$\Delta_g[k] = \begin{bmatrix} x_g \\ y_g \end{bmatrix} - \begin{bmatrix} x[k] \\ y[k] \end{bmatrix}.$$

Thus, $d_g[k]$ is the projected distance of $\Delta_g[k]$ onto the unit vector v_g. In order to reach the goal location $[x_g,\ y_g]$, we want $d_g[k]$ to approach 0 asymptotically.

Task 10 (2p)**.** *Assume that $\theta[k]$ equals to θ_g, i.e., the robot is correctly headed. Derive the equation to show how $d_g[k]$ changes with time. What is the maximal value of K_ω for $d_g[k]$ to converge to 0? Prove your statement formally. How do you choose K_ω?*

Task 11 (2p)**.** *Implement the controller and evaluate the performance by simulation results when you set $\omega = 0$. Is it possible to arrive at $[x_g,\ y_g]$ exactly? Why?*

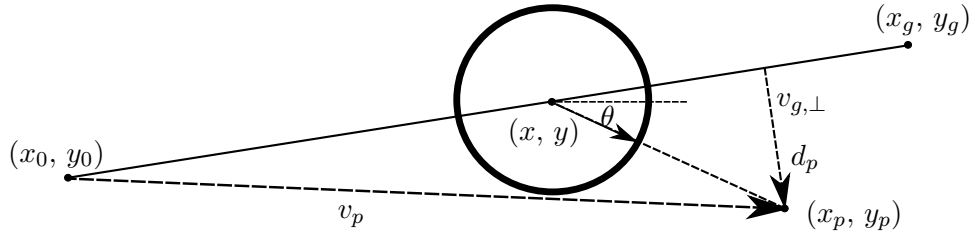

Figure 7: d_p as the inner product between $v_{g,\perp}$ and $v_p[k]$.

Regarding part (II), design a proportional controller

$$\omega[k] = K_\Psi\, d_p[k]; \tag{8}$$

where $d_p[k] = (v_{g,\perp}^T)\,(v_p[k])$ and $v_{g,\perp} = [\sin(\theta_g),\ -\cos(\theta_g)]^T$, $v_p[k] = \big[x[k]+p\cdot\cos(\theta[k])-x_0,\ y[k]+p\cdot\sin(\theta[k])-y_0\big]^T$, where $p>0$ is a design parameter.
The intuition behind this controller is that the robot always faces a virtual point $(x_p,\ y_p)$ with distance p ahead of it. The robot would go to this point, but in order to follow the straight line connecting $[x_0,\ y_0]$ and $[x_g,\ y_g]$ the proportional controller is used to change the robots orientation so that the virtual point is close to the straight line. Therefore the controller is used to minimize the distance of the virtual point to the line, which is $d_p[k]$ (see Figure 7).
(*Remark:* Searching for *Line follower* on YouTube might strengthen your understanding of the line-following part of the controller.)

Task 12 (2p)**.** *Assume that the robot lies on the straight line connecting* $[x_0,\ y_0]$ *and* $[x_g,\ y_g]$*. Derive the equation to show how* $d_p[k]$ *changes with time. What is the maximal value of* K_Ψ *for* $d_p[k]$ *to converge to* 0*? Prove your statement formally. (*Hint*: Approximate* $d_p[k] \approx p\cdot(\theta_g - \theta[k])$ *by assuming* $\theta[k]$ *is close to* θ_g*.)*

Task 13 (2p)**.** *How does the value of* p *influence the robot's ability to follow a line?*

Task 14 (2p)**.** *Implement the controller and show the control performance by simulation results when you set* $v=0$*. Is it possible to maintain* $d_p[k]$ *at* 0 *exactly? Why? (*Hint*: When implementing the controller, remember to use the actual* $d_p[k]$ *from* (8)*, instead of the approximated one.)*

Assume that you have obtained the two controllers from above. Now you want to combine them in order to achieve the line following objectives mentioned earlier. Note that ω and v are controlled separately by (7) and (8).

Task 15 (2p)**.** *Evaluate the control performance when both controllers are enabled through simulations. Plot the errors* $d_g[k]$ *and* $d_p[k]$*. How are they different from the case when only one controller is enabled?*

4.3.3 Hybrid Controller

Now you can combine the rotation and line following controllers into one hybrid controller that can navigate the robot from any initial state to any goal position. The idea is that the robot first changes its orientation by using the rotation controller so that it faces towards the goal location; then, it moves forward by using the line following controller in away that it reaches the goal location.

Task 16 (2p)**.** *Construct the hybrid controller that navigates the robot from an initial state* $[x_0,\ y_0,\ \theta_0]$ *to a goal position* $[x_g,\ y_g]$*. How do you formulate the guards from one discrete state to another? Model formally the controlled robot as a hybrid automaton:*

$$H = (Q, X, Init, f, D, E, G, R). \tag{9}$$

Task 17 (2p)**.** *Implement the hybrid controller and show the control performance by simulation results. Plot out both the continuous and the discrete state trajectories. Does the discrete state evolve as you expected? Motivate your answers.*

Here, the last of the three files, `RenewControllerState.c`, comes into play. You need to reset the controller state back to the orientation state, when you want to send the robot to a new goal position by pressing the `Set Reference` button. Keep in mind that you not only have to choose a new goal position but also update the start position.

Task 18 (2p)**.** *Execute the plan that you have proposed in Task 3 on the simulator by feeding the controller a list of waypoints (that is, the centers of the regions of the path that you derived in Task 3). Plot the results of a single execution of the plan as a trajectory of the robot in the xy plane, and comment on the observed performance of the robot.*

Task 19 (1p)**.** *Elaborate on how the performance of the continuous controllers affects the* safety *property of each transition (i.e., the desired property that the robot, during a transition between two regions, should always stay in these two regions.)*

5 Lab Session

In this part, you will use the controllers that you have designed in the previous part to control a real differential-drive robot, shown in Figure 8, in the Smart Mobility Lab of the Automatic Control Department [4].

Remember that you have to bring your working code that you have tested in the online simulator. You are not allowed to start the lab session if you do not have a working code ready to test. By *working code* we mean that, in the simulator, the robot is able to navigate from one point to another, while staying close to a straight line connecting the two points. Also, the robot should be able to recover from small displacements, as illustrated in the video tutorial. Finally, the robot should be able to transit between two region in the transition system while maintaining the property that it should always stay in these two regions.

If you experience problems that you cannot solve by yourself, you are welcome to seek help on the usual channels: write a post on CANVAS, or write an email to the appropriate TA.

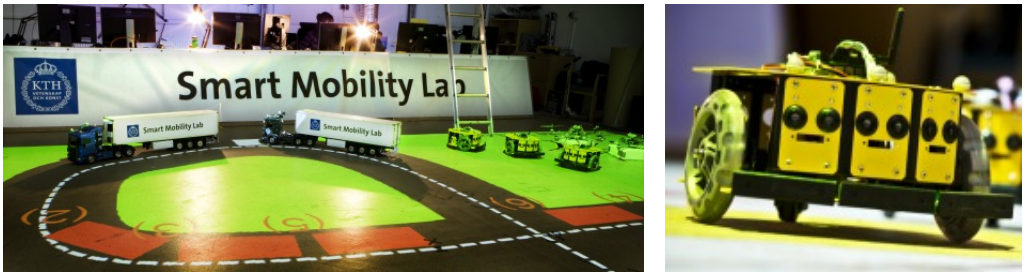

Figure 8: Left: the smart mobility lab at Automatic Control Department; Right: the Nexus differential-driven robot.

We will first let you control the robot manually for a few minutes; then, you will test the performance of the hybrid controller for navigation on the microcontroller by choosing different goal locations and following a predefined path.

5.1 Manual Control

To begin with, make sure `Activate Simulation` is unchecked in the GUI and press the `MoCap On` button to connect to the Motion capture system, then press the `Control On` button to be able to control the robot.

Task 20. *Try to manually navigate the robot to a goal position you have chosen. Is it difficult to reach a certain accuracy? Why?*

5.2 Controller Test

Once you have uploaded your code to the Arduino board, you can turn up the `Automatic Control On` and remotely send the next goal state. Then the robot will autonomously navigate to that goal state. If you are not satisfied with the performance, you may change/tune your controller and re-upload the program.

Task 21. *Choose different goal states and evaluate the control performance. Is it different from your simulation results? Why?*

Task 22. *Execute the plan that you have proposed in Task 3 by feeding the controller a list of waypoints (that is, the centers of the regions of the path that you derived in Task 3). Does the robot execute successfully the plan?*

References

[1] `http://arduino.cc`

[2] S. M. LaValle. Planning algorithms. *Cambridge University Press*, 2006.

[3] `http://arduino.cc/en/guide/Environment`

[4] `http://www.kth.se/en/ees/forskning/strategiska-forskningsomraden/intelligenta-transportsystem/smart-mobility-lab-1.415239`