EL2450
Hybrid and Embedded Control Systems
Homework 3

[To be handed in **February 24**]

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 5 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. The lab is done in the same groups as the report and all group members need to be present. You are only allowed to do the lab if you got of the report at least 40% of the points ($\geq$ 18 points) in the initial submission of the report. The grading of Homework 3 is as follows:

- *Pass:* $\geq$ 36 points for the report and lab session passed
- *Resubmission required:* $\geq$ 18 points for the report
- *Fail:* $<$ 18 points for the report of lab session failed

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 announcement *HW3 Lab session* 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, 14, 13, 15, 17, 18, 19, 20, 21, 22).

Please do your best to stay within a page limit of 15 pages. If you exceed this limit significantly, we might 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. **Nana Wang, nanaw@kth.se**
2. **Zifan Wang, zifanw@kth.se**
3. **Tommaso Zaccherini, tommasoz@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

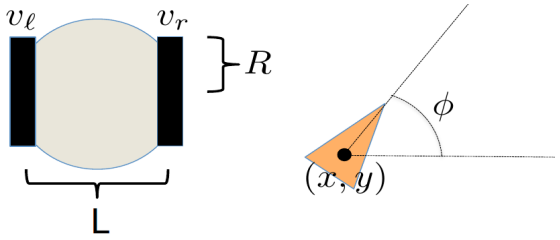

Figure 1: Differential-drive robot

files: one pdf file named `name1-name2-name3-name4-HW3.pdf` and three controller files `OwnVariables.c`, `Controller.c` and `RenewControllerState.c`.

3 Setup Description

3.1 Setup in the Lab

The robot used for this homework is a two-wheeled differential-drive TurtleBot3 Burger robot (see Figure 1). The TurtleBot3 is equipped with a Raspberry Pi and an OpenCR1.0 board to control the motors and access the onboard sensors, and with a wireless transreceiver to exchange information.

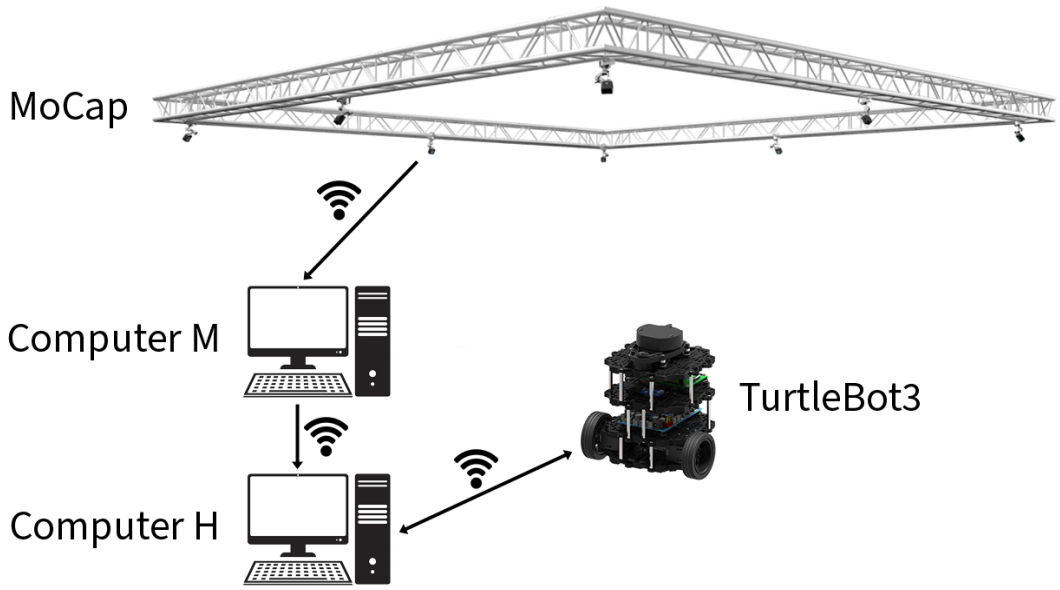

Figure 2: The hardware setup

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 centimeters; `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,
- obtain and record the robot's position data, and the control input
- specify the start and goal states for the robot.

It is important to note that the robot's start and goal positions are specified in meters in the GUI which is in contrast to the units used in the C-code.

The TurtleBot3 program will receive different messages wirelessly from the computer `H` (controlled by you through the GUI), and it will save them in different variables. For instance, it receives:

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

Therefore, while in the GUI the robots state is expressed in meters and degrees, the robot controller uses centimeters and degrees. Be careful to convert the measurements appropriately in the controller.

Most of the code that you need to run on the TurtleBot3 is already provided for you. In the course of the homework, you have to fill in the gaps in the files `OwnVariables`, `Controller`, and `RenewControllerState` using C-code.

3.2 Setup for the simulations

To make it easy to test your controller, we have provided an simulator of the real robot. We will provide a tutorial that illustrates how to use the simulator. The tutorial will be given on CANVAS. The simulator has the same GUI that you will find on the computer

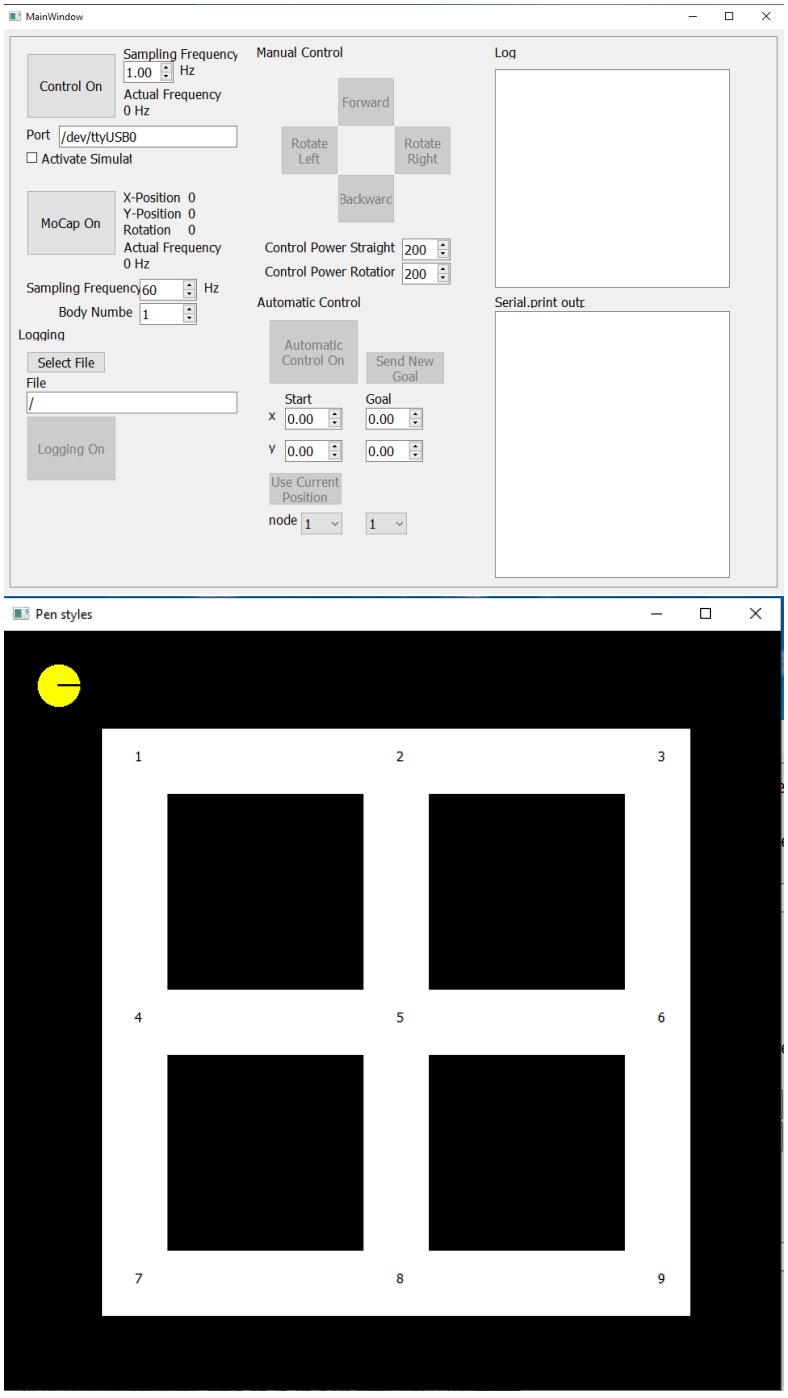

Figure 3: Top: the GUI. Bottom: The virtual corridor environment.

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; hence you can test the performance of controller in the online simulator. The files which you have to provide are the same as the labs, specifically as `OwnVariables.c`, `Controller.c`, and `RenewControllerState.c`. Each of these files must contain the code that you would place in the corresponding gap. After you have tested your controller on the simulator, you can simply copy-paste it on the real robot during the lab session.

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. 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. In tasks that ask you to evaluate the performance of your controller using simulations, you need to provide plots with simulation results that support your answer.

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` (see file `main.cpp`). Note that θ is measured 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 $3 \times 3\text{m}^2$.

The idea is that the robot moves from the center of one region to the center of a neighboring region by either moving up, down, to the left or to the right. In order to obtain a meaningful discretization of the workspace, the robot shall be completely contained within one region once it reaches its center, and only be in at most two regions simultaneously during the transition. We assume that the maximum size of the robot is $d_R = 37.5\text{cm}$, i.e., the robot does not occupy an area larger than $37.5\text{cm} \times 37.5\text{cm}$.

Moreover, we consider a set of properties of interest (atomic propositions) that hold in some of the regions of the workspace. Let these be properties "red", "blue", "green" and "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 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 positions $(-0.75, 1.4)$, $(0.9, 0.9)$, $(0.3, 0.2)$, $(0.25, -0.75)$, $(1.2, -1.4)$,
- "green" holds at positions $(-1.23, 1.25)$, $(1.25, 1.25)$, $(-0.9, 0.2)$, $(-1.2, -0.7)$, $(0.6, -1.2)$.

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

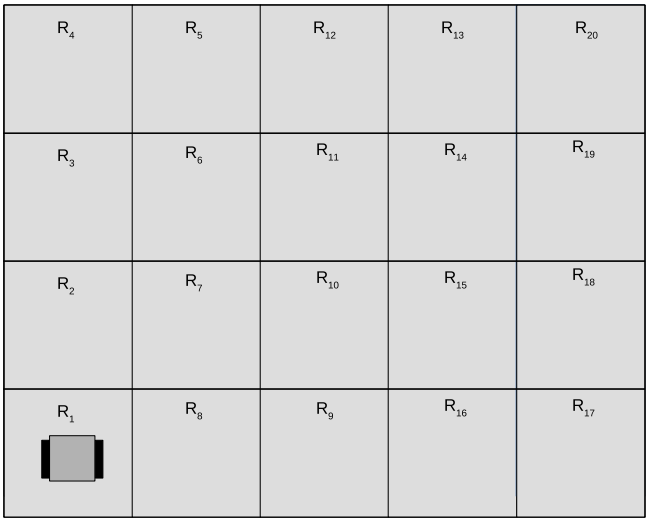

Figure 4: Workspace discretization in rectangular regions

Task 2 (8p)**.** *Formulate a labelled transition system* $\mathcal{T} = (S, S_0, \Sigma, \rightarrow, AP, \mathcal{L})$ *that captures the aforementioned specifications by following the four subsequent steps:*

a) Provide a graphical illustration of the discretized workspace similar to Figure 4 but use a discretization that yields $K = 36$ *regions. Number the regions analogously to Figure 4. Indicate where and which type(s) of the atomic propositions are true. (2P)*

b) State the size of one region as a tuple (d_x, d_y) *where* d_x *and* d_y *denote width and height of a region, respectively. (1P)*

c) Discuss the advantages and disadvantages of the chosen number of states. Would a finer or a rougher discretization be better suited than the chosen discretization? (2P)

d) Formulate the labelled transition system $\mathcal{T} = (S, S_0, \Sigma, \rightarrow, AP, \mathcal{L})$*. (3P)*
Hint: Consider region R_i *and its center* $c_i = (c_{x,i}, c_{y,i})$*. The neighboring regions of* R_i *are given as* $\mathcal{N}_i = \{R_j \in S \,|\, (c_{x,j}, c_{y,j}) = (c_{x,i} \pm d_x, c_{y,i}) \text{ or } (c_{x,j}, c_{y,j}) = (c_{x,i}, c_{y,i} \pm d_y)\}$ *where* d_x*,* d_y *state the size of one region as determined in Task 2b).*

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 (as a next state).*

- *The robot should never go to "obstacle" regions.*

Find a path, as an infinite sequence of states of the Transition System, that satisfies the aforementioned behavior. The infinite sequence shall be written as a finite prefix followed by a finite suffix, where the latter is run infinitely many times. This can be marked by using brackets followed by an asterisk written as $(\texttt{SUFFIX})^*$.

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. In order to ensure that the robot does not pass through regions different from the current one and the target region during the transition since these might contain obstacles, a controller shall be designed that drives the robot from one region directly to the center of the target 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++. If you have not worked with C/C++ so far, you find a brief introduction `CBasics.pdf` on Canvas. All you need to know about C/C++ for this homework is summarized here.

You can implement the controller by filling in the three files `OwnVariables`, `Controller`, and `RenewControllerState`. These files are uploaded to the online simulator to test your controller. For the real robot in the lab session, you need to use copy-and-paste. It is highly recommended that you implement and extensively test each part of your controller before designing the subsequent parts in order to facilitate debugging.

Those variables that are already defined inside the provided code are listed in Table 1. All additional variables that you introduce yourself have to be declared in the file `OwnVariables.c`. Note that all predefined variables in Table 1 are constants except for the variables *left* and *right*. Constants do not change their values at any time.

The second file is `Controller.c`. Here you implement everything which belongs to the controller itself. Do not declare any variables here. Here, you calculate v and ω to determine u_l and u_r. Later, you will decide in which controller state the controller has to be.

The last file is `RenewControllerState.c`. It is used 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. At first, we design a rotation controller that aligns the robot with the line connecting the robot to the center of the target region. Thereafter, we design a *go-to-goal* controller, that actually drives the robot to the center of the target region.

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

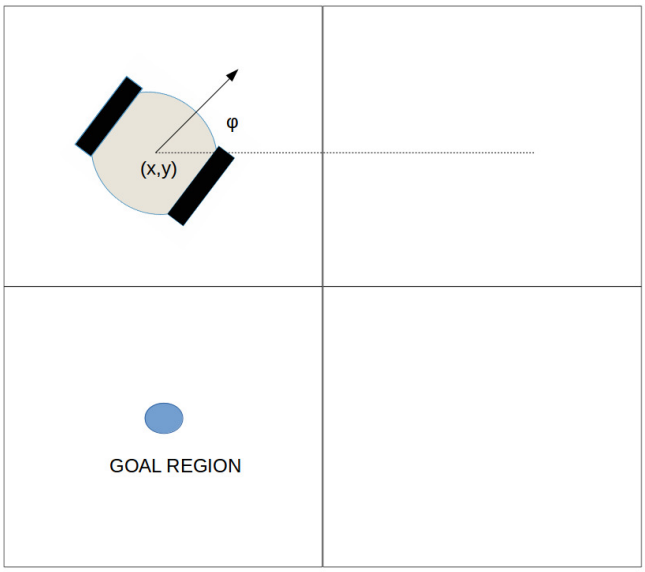

Figure 5: The mobile Robot and the goal region for a potential transition.

Task 4 (1p)**.** *Explain how this choice of the hybrid control strategy ensures that the robot does not enter any other region apart from the region, where it starts, and the target region. Would it be better to simultaneously control both the relative orientation and distance of the robot to the goal region? Motivate your answer.*
Hint: *Have a look at Figure 5 and take the size of each region into account.*

4.3.1 Discrete-Time Rotation Control

The objective of the rotation controller consists of two parts:

(I) Controlling the rotation velocity ω changes the robot's orientation θ such that it is close to a desired angle θ^R;

(II) Controlling the translation velocity v maintains the robot's position $[x,\, y]$ close to its initial position $[x_0,\, y_0]$ when it starts rotating (constant position).

Note that v and ω are defined in (2). With respect to Part (I), design a proportional discrete-time controller

$$\omega[k] = K_{\Psi,1}(\theta^R - \theta[k]); \tag{5}$$

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

Task 5 (2p)**.** *Let the sampling time be denoted by h. For which values of $K_{\Psi,1}$ is θ asymptotically stabilized in θ^R? Prove your statement formally. How do you choose $K_{\Psi,1}$? Motivate your choice.*

The online simulator uses sampling time $h = 1s$. During the lab session, however, you need to select a reasonable sampling time yourself. Therefore it is important that you specify and implement all control gains in terms of h such that you can easily adjust the sampling time.

Task 6 (2p)**.** *Implement the controller. Set $v = 0$ and simulate your controller using the online simulator. Show the simulation results and comment on its performance. Does the designed controller asymptotically stabilize θ in θ^R? Justify your answer.*

Regarding Part (II), design a proportional controller

$$v[k] = K_{\omega,1}\, d_0[k]; \tag{6}$$

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 that $d_0[k]$ asymptotically converges to 0.

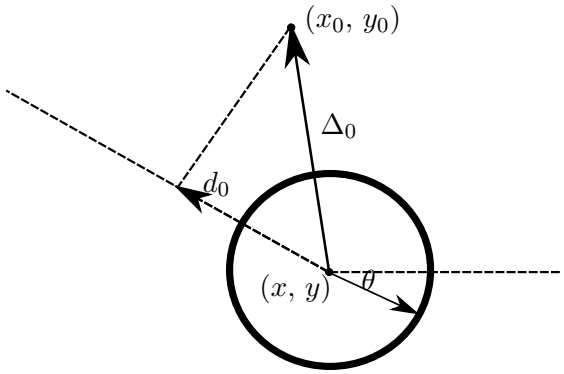

Figure 6: d_0 as the inner product between Δ_0 and $v_c[k]$.

Task 7 (2p)**.** *Assume that $\theta[k+1] = \theta[k]$, i.e., θ does not change. Derive the difference equation that describes the discrete-time dynamics of $d_0[k]$. For which values of $K_{\omega,1}$ is d_0 asymptotically stabilized in 0? Prove your statement formally. How do you choose $K_{\omega,1}$? Motivate your answer.*

Task 8 (2p)**.** *Implement the controller. Then, set $\omega = 0$, simulate your controller using the online simulator and demonstrate its performance by showing simulation results. Is d_0 asymptotically stabilized in 0? Is it possible to ensure that $[x[k],\ y[k]]^T$ stays exactly at $[x_0,\ y_0]^T$ for any initial orientation? Explain your observations.*
Hint: *Keep 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 to the case when only one controller is enabled? Explain your observations.*

4.3.2 Go-To-Goal Controller

When the robot converged (approximately) to desired orientation θ_g, it shall move towards the desired goal point (x_g, y_g), which is the center of the goal region. To avoid that the robot significantly deviates from the line that connects the robot's center of mass with (x_g, y_g), we design a line-following algorithm that always keeps the robot close to this line. The line following controller 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 $[x_g,\ y_g]$.

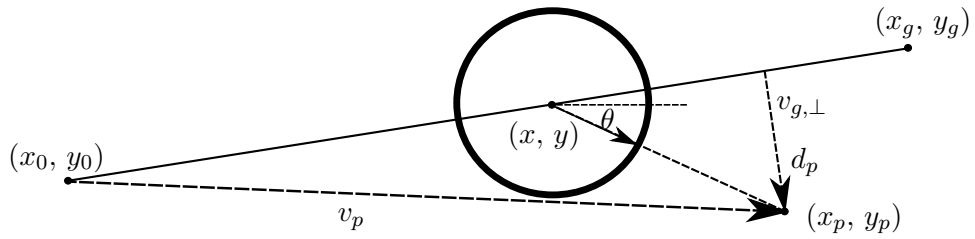

Figure 7: d_p as the inner product between $v_{g,\perp}$ and $v_p[k]$.

Regarding part (I), design a proportional controller

$$v[k] = K_{\omega,2}\, d_g[k]; \tag{7}$$

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 that $d_g[k]$ asymptotically converges to 0.

Task 10 (2p)**.** *Assume that $\theta[k]$ equals to θ_g, i.e., the robot is correctly oriented. Derive the difference equation that describes the discrete-time dynamics of $d_g[k]$. For which values of $K_{\omega,2}$ is $d_g[k]$ asymptotically stabilized in 0? Prove your statement formally. How do you choose $K_{\omega,2}$? Motivate your answer.*

Task 11 (2p)**.** *Implement the controller. Then, set $\omega = 0$, simulate your controller using the online simulator and demonstrate its performance by showing simulation results. Is d_g asymptotically stabilized in 0? Does $[x[k],\ y[k]]^T$ asymptotically converge to $[x_g,\ y_g]^T$ for any initial orientation? Explain your observations.*

Regarding part (II), we design a proportional controller

$$\omega[k] = K_{\Psi,2}\, d_p[k]; \tag{8}$$

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 proportional controller is used to minimize the distance of the virtual point to the line connecting (x_0, y_0) and (x_g, y_g).

Task 12 (2p)**.** *Let the robot be located on the line connecting (x_0, y_0) and (x_g, y_g). Furthermore, we assume that θ is close to θ_g such that we can approximate d_p as $d_p[k] \approx p\cdot(\theta_g - \theta[k])$. Derive the difference equation that describes the discrete-time dynamics of $d_p[k]$. For which values of $K_{\Psi,2}$ is d_p asymptotically stabilized in 0? Prove your statement formally.*

Task 13 (2p)**.** *Implement your controller. Then, set $v = 0$, simulate your controller and demonstrate its performance for some control parameters p and $K_{\Psi,2}$ by showing simulation results. Does d_p converge asymptotically to 0 or does a steady state error remain? Explain your observation.*
Hint*: When implementing the controller, remember to use the actual definition of d_p from (8) instead of its approximation.*

Task 14 (2p)**.** *Set again $v = 0$. By using simulations, explain how the value of p influences the robot's ability to follow a line. Based on this, reconsider the chosen values for p and $K_{\Psi,2}$ and propose an improved choice.*
Hint: *To analyze the impact of p on the control performance, select $K_{\Psi,2}$ such that the pole of the (linearized) closed-loop system is invariant with respect to variations in p. Then, simulate the your controller for different values of p to develop an intuition. In the simulation, it might be also interesting to position the robot next to the line connecting (x_0, y_0) and (x_g, y_g) with some displacement.*

Now we combine the two controllers obtained for Part (I) and (II) in order to obtain a line following controller. The inputs ω and v are controlled separately by (7) and (8).

Task 15 (2p)**.** *By using simulations, evaluate the control performance when both controllers are enabled. Plot d_g and d_p. How are they different from the case when only one controller is enabled?*

4.3.3 Hybrid Controller

Now we 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, the robot moves forward by using the line following controller such 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 choose the discrete states? And how do you formulate the guards for each transition? Formally model 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 demonstrate the control performance by using simulation results. Plot both the continuous and the discrete state trajectories. Does the discrete state evolve as you expected? Motivate your answer.*

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. In order to update the goal position, call the function `send_done()`.

Task 18 (2p)**.** *Execute the plan that you have proposed in Task 3 on the online simulator by using a list of waypoints (these are 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.

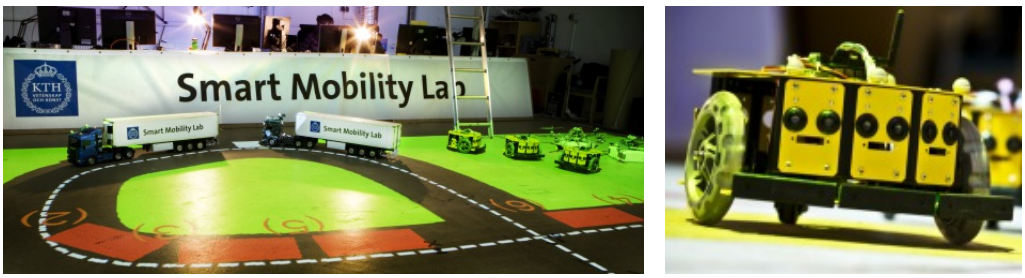

Figure 8: Left: the smart mobility lab at Automatic Control Department; Right: the Nexus differential-driven robot.

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 TurtleBot3 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? Try different sampling times in order to improve the observed control performance.*

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 successfully execute 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`