RoboLab Real-time Rescue

Lab Compendium for the Course TDDD07 Real-time Systems

Mikael Asplund, Eriks Zaharans, Simin Nadjm-Tehrani, Klervie Toczé,
Rodrigo Saar de Moraes, Chih-Yuan Lin

October 2023

Contents

1 Introduction

Welcome to the Robolab Rescue lab series for TDDD07 Real-time Systems! This document is intended to help you to successfully complete these labs and thereby acquire new worthwhile knowledge and some hands-on experience. Please read this document carefully in good time before attending the preparatory lesson for the lab. It contains information regarding steps you need to complete before the first scheduled lab occasion.

The document is composed of four sections. We begin with a birds-eye perspective on the lab environment and what you will learn and do during the lab. Section 2 describes the necessary information you need to know about the physical lab environment that you will be working with. Section 3 provides detailed instructions on how to prepare for and what to do during the scheduled lab sessions. Finally, Section 4 gives information regarding how you will be examined on having completed the steps of the entire laboratory course.

The appendices contain auxiliary information which you will make use of during the labs (including detailed instructions on how to handle the robot and how to start the system), but which you do not need to read beforehand.

1.1 Overview and goals

The goal of this lab series is that you will acquire new insights and get hands-on experience in building a distributed real-time system with multiple hardware and software components. After completing this lab course you should be able to analyse timing properties for a complex system and identify important factors that affect the ability of a subsystem to contribute to timeliness. During the labs you should learn how to structure and evaluate a program which is composed of a number of real-time tasks and which communicates with other components in a distributed system with resource-constrained communication channels.

Moreover, this lab series is intended to give you an opportunity to practice documenting and presenting your technical solutions in written and oral form. You should be able to present problems you have encountered and to justify your solutions to these problems. Finally, during the second part of the lab you will need to coordinate your solution with other groups which will strengthen your ability to plan your work to fit in a larger context.

1.2 Scenario

We live in a fragile society where disruptive events such as storms, earthquakes, tsunamis, or large-scale fires can cause major loss of human life and destroy crucial infrastructure on a large scale. In the aftermath of such events, volunteers, professional personnel and international relief organisations all contribute to saving lives and work to restore some sort of societal fabric. Autonomous and semi-autonomous robotic systems can play an important role in the immediate post-disaster phase. Tasks like searching for and rescuing persons trapped in dangerous and inaccessible areas, surveying damaged

areas, and clearing out debris can be performed by teams of ground-based and flying vehicles.

In this lab you will be working with an emulated version of a team of search and rescue robots. In this scenario, a set of ground-based robots will search for victims that are to be reported to a mission control centre. The area to cover in the search is larger than what one robot can cover alone in short enough time. Therefore, the robots will need to collaborate and share information with each other about their whereabouts and which parts of the disaster area that have been covered. To make an effective rescue mission even in presence of unexpected terrain related events the robots need to be able to manoeuvre semi-autonomously from the mission control central.

1.3 Lab assignments in brief

Students work pairwise in a group and will be responsible for programming one robot. There are three lab assignments.

Lab 1: Your job is to schedule recurrent instances of a set of tasks running on the robot platform. You first need to analyse the tasks with respect to their worst case execution time, internal dependencies and other factors which affects how often each task can be run. You will need to find the limits for how often/seldom a task can be run and based on that information design a suitable task schedule. After implementing this schedule you will evaluate your system with respect to timeliness and assess the resulting mission performance.

Lab 2: You will also need to schedule the wireless communication between the robots. Your job will be to ensure that all transmissions originating from your robot happen within predefined time slots. The amount of data transmissions generated at the robot exceeds the data that can be sent during the allocated transmission slots. Therefore, you will need to prioritise in which order messages are to be sent as well as determining how much data will fit within the allowed slot.

Lab 3: The final lab is to analyse and improve the dependability aspects of the system. The lab gives bonus points towards the exam and you should attempt it only after demonstrating lab 1 and lab 2 within given time frames. The assignment is to describe fault models for the system and suggest changes that would allow detecting and managing these faults.

2 Lab environment

An overview of the lab environment is illustrated in Figure 1. The system consists of a number of robots (one for each pair of students) which operate in a closed area, a number of workstations (one for each pair of students), tags which represent victims, and a dedicated WiFi router which enables wireless communication between the robots and the workstations.

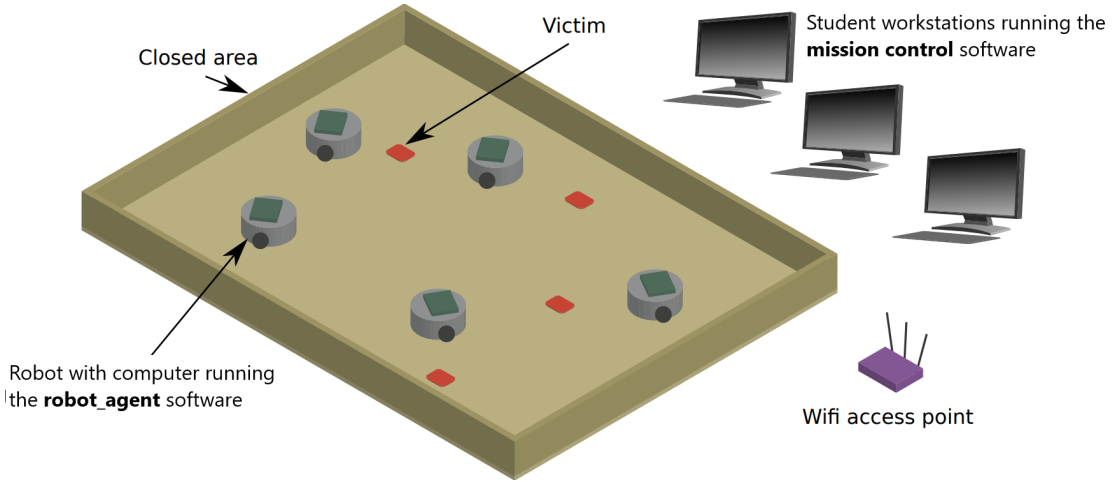

Figure 1: Lab environment overview

2.1 Physical setup

We will now proceed to give a brief overview of each of the physical components of the system.

2.1.1 Robots

The robot are composed of three main components, the robot chassi, a small mounted computer (Raspberry Pi), and a mounted RFID reader. The computers connect to the robot chassi through USB as shown in Figure 2.

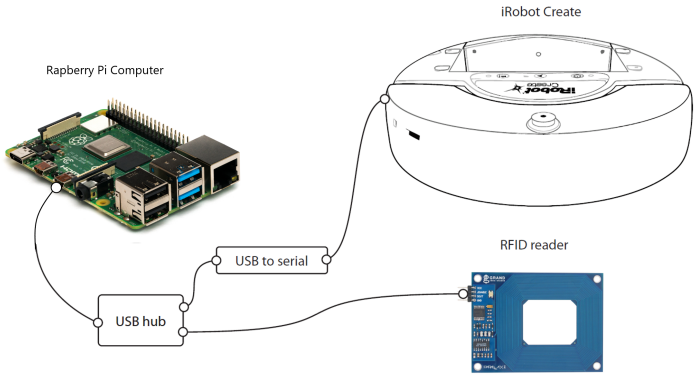

Figure 2: Robot platform setup.

The robot chassi is an iRobot Create, which is a robotics development platform intended for educational use. The robot is equipped with two-wheel differential drive (heading direction is changed by changing relative rate of rotation of the wheels), and a

number of sensors. The sensors (including bump sensor and rotary wheel encoders) allow the robot to avoid getting stuck in corners and to get a sense of relative movements. However, in order to determine its location in the field, additional sensors are needed. Controlling the movement of the robot and reading sensor information can be done using the Open Interface standard over a serial port (to which the computer is connected via USB). It is important that you handle the robots carefully. If the robots are out of control, turn them off using the power button, and if for some reason that does not help, pull out the battery (see instructions in Appendix C).

The equipment mounted on top of the robot chassi, are composed of a Raspberry Pi (RP) 4 model B with 4GB RAM, a PiJuice HAT which together with a battery acts as an uninterruptible power supply for the RP. They will be running the robot agent code which controls the robots, the RFID reader and communication. They interface with the robots through a USB connection and with the infrastructure through WiFi. The computers should only be connected to the lab WiFi (otherwise the system will not be under our control any more). This is the default setup, do not change it to use other routers, such as Eduroam.

The RFID readers are mounted in the robot enclosures beneath the computer and provide the capability to sense RFID tags which are placed on the ground. This is used to detect victims as well as to help determine the position of the robot within the field.

2.1.2 Field and victims

The robots operate in a closed area measuring 6x4 meters. The field is covered by a thin plywood layer under which 320 RFID tags have been placed. These tags are used by the robot localisation algorithm to improve the estimated location accuracy by each robot. The location of each of the localisation tags are known beforehand by the system. When such a tag is read, the localisation algorithm can deduce that the robot is very likely within a few centimetres from the location where the tag is located. Note that the robot is not necessarily positioned exactly over the tag, and that there can be a delay from when the tag was read by the RFID reader to when it is registered by the localisation algorithm.

In addition to the localisation tags, there is also a number of tags which represent victims. The locations of these victim tags are *not* known beforehand but should be found by the robots as part of the search and rescue mission.

2.1.3 Surrounding infrastructure

The robots are able to communicate with each other as well as with the mission control central using a WiFi access point (AP) which is operating on the 2.4GHz band. The AP is intentionally not connected to the Internet. The infrastructure is dimensioned so that up to eight robots can operate on the field at the same time.

The workstations provide the development environment where you will program and access information from online sources. This is also where the mission control software will run allowing you to control the robot mission and receive information updates.

The workstations are part of the IDA undergraduate computer network, so you will have access to your home directory as well as Internet connectivity. Moreover, the workstations have a wireless interface which means that they can connect to the WiFi AP and to the robots.

2.2 Software

As you probably have realised, this system is composed of a significant number of processors ranging from basic embedded micro controllers to a quad-core processor. Each processor will be running some piece of software (in some cases proprietary) where timeliness is a factor. We now focus on the robot agent software running on the RP which is where the high-level robot control takes place and where your code will be running. Towards the end of this section we also explain how the mission control software (running at the workstation) operates, but since you will only use this software, not modify it, we keep this discussion very brief.

2.2.1 Robot agent software

The robot agent software is responsible for making the robot operate according to its specified mission. This software runs on the robot computer and uses the serial interface to the robot chassi to query sensors and send commands to control the movement of the robot. The robot agent is written in the C programming language and runs as a single process on the Raspberry Pi OS (formerly Raspbian). Figure 3 shows the software architecture of the robot agent.

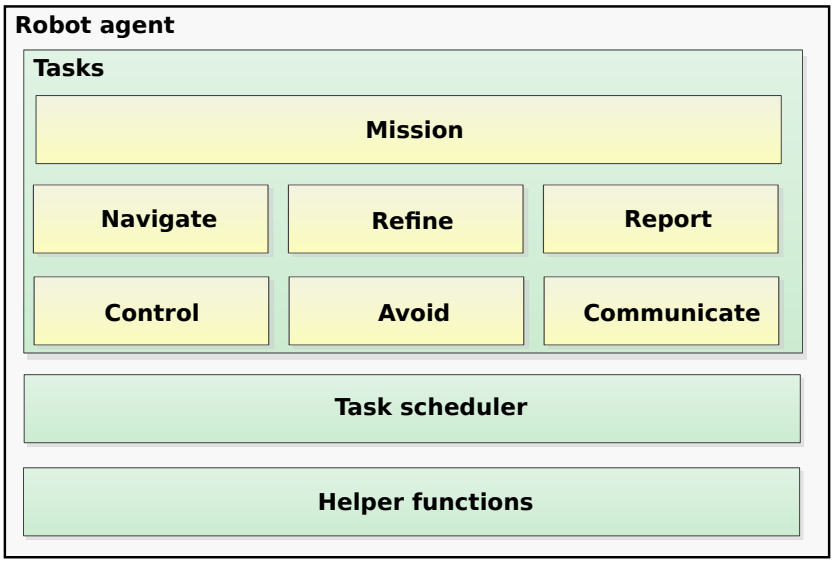

Figure 3: Robot agent software

Tasks The tasks in the robot agent are pieces of code which each have a specific purpose and some temporal properties and requirements. We provide a high-level overview here, and refer to Appendix E for a more detailed description with pseudocode for each task.

The *Mission* task manages the overall mission of the robot which is essentially to search for victims and report found victims to the mission control centre. In particular, the mission task is responsible for keeping track of reported victims, and starting and stopping the other tasks. Stopping the tasks can for example occur because a stop message is received from the mission control. Mission control also generates and sends a stream data to Communicate task (right now stream data is just a randomly generated data, but in future it will be a video stream generated by a camera). Stream data is generated in a constant rate.

The *Navigate* task is responsible for determining the direction in which the robot should go. The navigation logic is based on a pheromone mobility model where each robot tracks its movement in a virtual pheromone map, similar to the way ants use pheromones on the ground to tell other ants where they have been. In the case of search and rescue, the existence of pheromones on the map will make the other robots avoid that area since it has already been covered. As time elapses the strength of the virtual pheromones decreases, allowing the area to be searched anew.

The *Refine* task uses the RFID reader to refine the robot positioning using the tags below the wooden surface that have known positions. The task queries the RFID reader for tags and checks whether the tag is a localisation tag. If the tag is recorded in the localisation database, then the particle filter is updated with this new information. The role of the particle filter is to combine the odometry location data available from the robot (see Control task below) with the RFID tag data through sensor fusion. If a tag is found which is not a localisation tag, then this represents a victim being found, so this information is forwarded to the Report task.

The *Report* task receives information about a found victim, it checks whether that victim had already previously been reported. If not, then victim information is given to mission task for victim registration and a message is sent to the Communicate task to send this information to the mission control centre (and display it on the operator screen).

The *Control* task is responsible for low-level communication with the robot. First the task reads the odometry (wheel encoder) sensor data from the robot and incorporates this data in the localisation module (particle filter). Then it send the motor commands to the robot chassi with direction decided by the Navigate task.

The *Avoid* task is responsible for making the robot avoid getting stuck when running into obstacles. If the robot tries to go forward where it cannot, there is a high

risk of the motors being damaged. It is possible to detect this situation using the "bump" sensor in front of the robot. The avoid task queries the bump sensor, and if it detects that the bump sensor is pressed, then it instructs the robot to turn to another direction. As robot can execute only one motor command, running Avoid task directly after Control task will override command sent by the Control task. This also applies to inverse order: Avoid tasks motor command can be overridden by Control task.

Finally, the *Communicate* task manages all wireless communication by reading incoming messages and transmitting the messages generated by the other tasks in the system. Since the actual sending of messages occurs at lower layers in the communication stack, the communicate task simply puts the messages on a queue without waiting for the messages to be transmitted.

Task scheduler The task instances are dispatched by the task scheduler which determines the order in which tasks are run and how long to wait until the next task is run. Note that the robot agent code runs as a single Raspberry Pi process which means the task scheduler in the robot agent has no possibility to preempt currently running tasks.

Helper functions Basic functions like communicating with the robot chassi and the RFID reader are implemented as helper functions. These encapsulate some of the low-level implementation-specific details, allowing the a more abstract implementation of the tasks. Some of these helper functions will be useful for your solution, and these will be described with a basic API description on the course web.

2.2.2 Mission control software

In order to control the robots from a mission level perspective, there is a mission control software that you will run on your workstation. This software serves two purposes. First, it shows the whereabouts of the robots on the field as they report it. What information is shown is dependent on the quality of the schedule you implement in the robot agent software. Second, the mission control software is able to start and stop the robots as well as to employ a continuous safety mechanism which only allows the robots to proceed if they are able to regularly report their location. This latter feature will be used in lab 2 when you will implement a form of real-time communication.

3 Instructions for lab sessions

In this section we will explain the detailed steps for each of the three labs, and the system requirements you will need to meet. Detailed instructions on how to setup the system will be given in Appendix D, on the course web and by your lab assistant.

3.1 Preparations

Before coming to the first lab session there you need to prepare by carefully reading this compendium. You should also familiarise yourself with the documentation appendices in this report which contain important information on how to handle the robots (Appendix C) and how to get started with the lab skeleton (Appendix D). Before the first lab session, we will also give an introductory lesson including explaining some of the practical issues.

In order to make sure that you have completed the above steps before the lab session, you are required to answer the preparatory questions provided in Appendix A by sending an email to your lab supervisor. *Note, you need to do this at least three hours before your lab session starts.* When you have passed this step, the lab assistant will send you the necessary information to start the labs.

3.2 Lab 1

Lab 1 is concerned with scheduling the tasks of the Robot agent. Your job will be not only to find a working schedule that meets the requirements, but actually to explore the space of possible solutions and to implement, evaluate and document the solution which you think is best. The code you write in lab 1 will reside in the task scheduler (which is found in the file `scheduler.c`).

To complete lab 1 you should perform the following steps.

1. Measure the execution time of all tasks under varying conditions. Some suggestions: different execution periods/order of the tasks; robot in different situations: near walls, close to the victims, etc. For each task you should have at least 20 measurement points. Document your measurements carefully in the lab report.

2. Determine the minimum and maximum acceptable period for each task. Justify your results.

3. Design and implement a schedule for the tasks.

4. Evaluate and measure the performance of the system with regards to the timing requirements as well as the resulting localisation accuracy of reported victims.

In order to determine the acceptable period for each task and to create a suitable schedule you need some additional information. This can be acquired in some cases by analysing the system properties (see Section 2.2.1 and Appendix E) as well as the application requirements below, and in some cases by varying the period of the task and observing/measuring the resulting system performance.

Application requirements In addition to trying to achieve the best performance possible, there are some temporal system requirements that must be satisfied by your solution.

- The Avoid task must be run at least every 100-150ms to avoid overheating the motors.
- The time from when a victim is read by the RFID reader to when a message is sent must be less than 1300ms.
- If a stop command is received from the mission control, then the robot must stop within 1300ms.

3.3 Lab 2

So far we have assumed that there were no resource constraints on communication bandwidth, so Lab 1 focused on the CPU resource. In lab 2 you will be scheduling the wireless transmissions coming from the robot. For this lab we change the scenario slightly by adding a safety-feature in the mission task of the robot agent. This safety mechanism will ensure that a *go-ahead* message has been received from the server every second. If a go-ahead message is not received, then the robot will stop until eventually a new go-ahead message is received.

Moreover, the mission control server will enforce a time-division scheme on all incoming communications. This means that every robot will get a recurring time-slot in which it is allowed to send messages. The server only accepts messages within this time slot, and if messages are transmitted outside the time-slot, then no go-ahead message will be sent to the robot.

There are four types of messages that can be sent from the robot.

- **Victim reports** which are sent when a victim is found. It is critical that these messages are transmitted.
- **Location messages** which contain information about the position of the robot. A location message is generated every time the Control task runs. It is important that the latest location of the robot is conveyed to the mission control since otherwise the robot will not receive a go-ahead message.
- **Pheromone maps** which are shared among the robots to indicate which areas that have been searched. A complete map of the area is generated every time the Navigate task is run and the data is split in 8 map sectors. Old pheromone maps are redundant as the newest map also contains information about the old pheromone maps.
- **Stream data** which emulates live video transmissions and which should be transmitted if there is room, but can be considered as non-critical.

Although communication uses fast WiFi network, the communication between robot and Mission control is simulated with a much lower data rate - 153600 bits per second.

To complete lab 2 you should perform the following steps.

1. Make the communication task (located in the file `tasks/task_communicate.c`) run in the pre-defined time slot. Note that you might have to change your schedule from lab 1 (depending on how often you ran the communication task in lab 1). In any case you need to synchronise the schedule so that the communication task always runs in its slot.

2. Determine a priority order for the different message types.

3. Modify the communication task to prioritise messages and perform admission control so that no messages are sent outside the slot.

4. Test your solution together with one or more other robots connecting to the same mission control software. Since each robot should only use its own time slot for communication, you should be able to successfully run with up to eight robots (each with its own correct implementation) connecting the same mission control centre.

5. Measure how many location, pheromone and stream messages you are able to transfer from the robot to the mission control software without sacrificing the performance of localisation, search and victim reporting.

You should implement a solution where the robot makes as few stops as possible due to not receiving the go-ahead message[1]. You should also evaluate how much valuable but non-critical data you are able to transmit from the robot to the server.

3.4 Lab 3

The lab gives **bonus points** towards the exam and you should attempt it only after demonstrating lab 1 and lab 2 within given time frames.

So far the application did not explicitly include any fault models. In this lab you are required to use your application and platform knowledge (both robot and server side) to:

1. Describe plausible fault models for the system (at least two different ones).

2. Point out where in the code (server or robot) added fault detection and management code should be injected (describe as pseudocode). This part gives up to two bonus points.

3. Extend the code to implement your solution(s) and demonstrate the results. This part gives up to two additional bonus points.

The outcome of your work on lab 3 should be documented in a dedicated *individual* report.

[1] Typically packet loss (that can happen due to interference) should not block your robot from moving. The assistant will help you to see if you need extra work for other reasons.

4 Examination criteria

This section lists the detailed requirements for passing the course. In essence you are required to perform analysis, implement a solution, evaluate and document the entire process in a lab report.

4.1 Steps to complete

In order to *pass* the lab course you need to complete the following 7 steps.

1. **Lab 1 preparatory questions**, answer the preparatory questions for lab 1 contained in Appendix A.1 and send them by email to your lab assistant.
2. **Lab 1 demonstration**, demonstrate your working solution to the lab assistant.
3. **Lab 1 code submission**, send your solution code to the lab assistant.
4. **Lab 2 preparatory questions** answer the preparatory questions for lab 2 contained in Appendix A.2 and send them by email to your lab assistant.
5. **Lab 2 demonstration**, demonstrate your working solution to the lab assistant.
6. **Lab 2 code submission**, send your solution code to the lab assistant.
7. **Lab report**, write a lab report according to the requirements given in Appendix F and send it to your lab assistant through Urkund.

During the lab demonstrations, the assistant will check that you have performed the different lab steps (see Sections 3.2 and 3.3) as well as that you have all the necessary material for writing the report (see Appendix F).

When sending emails to the lab assistant the subject should be `TDDD07, Group X, Y for lab Z`, replacing `X` with your group (e.g., B2), `Y` with the topic (e.g., answers to preparatory questions), and `Z` with the lab number (1-3). The email-addresses will be available on the course web pages. The lab report should be sent to a special Urkund-address which will also be available on the course web page.

The deadline for each examination step is listed on the course web page. Please note that these deadlines are firm. This means that we will not consider late submissions after the provided deadlines. After the course has ended, there will be two more opportunities to pass the labs in connection with the retakes. Details of these deadlines will be announced on the course web page.

4.2 Rules of conduct

This lab course is intended to be an opportunity for you to learn and develop new skills. We believe that this is also the reason you are here.

The examination section of the course web pages lists the rules for examination of computer lab assignments at IDA. Make sure you have read and understood them. The

rules are also available at the following address: `http://www.ida.liu.se/edu/ugrad/lab_exam/index.en.shtml`

Appendix A Preparatory questions

A.1 Lab 1 preparatory questions

1. What can happen if the robot is stuck and motors are active?
2. What do you do if the robot is out of control?
3. What is the role of the Refine task?
4. How often does the Avoid task need to run?
5. Can tasks in the robot agent be preempted?
6. In which file(s) will solutions for lab 1 go?
7. What is a metric?
8. What metrics do you plan to use to show your WCET estimates are good enough?
9. How do you plan to evaluate and measure the performance of the system as defined in Appendix F?
10. When is the performance of the system satisfactory?

A.2 Lab 2 preparatory questions

1. What is the role of the go-ahead message?
2. What are the different message types?
3. In which file(s) will solutions for lab 2 go?
4. How can you make sure that the robot only sends messages within its allocated time slot?
5. What metric(s) do you plan to use in lab 2?
6. How do you plan to evaluate your solution?
7. When is your solution satisfactory?
8. How do you ensure that the performance you had in lab 1 does not degrade due to lab 2?

Appendix B Computer handling

Our computers are composed of a main board, a portable uninterruptible power supply (UPS), and a battery. There are some LED lights and buttons on the main board and UPS that convey important information. These lights and buttons will be introduced in this section.

The main board.

Fig 4 shows the main board, Raspberry Pi, of our computer to control the robot. There are two LED lights conveying information that could be helpful for problem diagnosis in this lab. The power LED turns red when the power is on and blinks when the power supply goes wrong with unstable current or low voltage. Since we have an UPS installed, the power LED should always be red. The activity LED turns green when the computer is busy working and blinks consistently when losing network connection or stopping in an emergency situation.

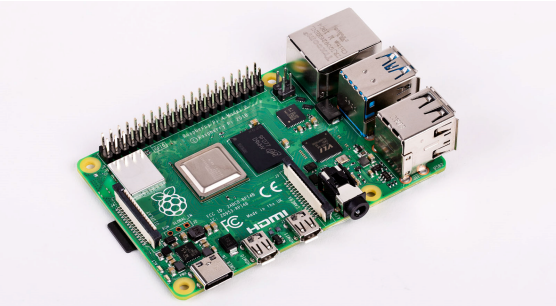

Figure 4: Raspberry Pi Computer Model B (4GB)

The UPS.

Fig 5 shows the UPS, PiJuice HAT, of our computer. There are two LED lights and two buttons. For this lab, we only need to know the power status light, LED1, and the power button, SW1/J5.

- **LED1** shows the power status. When the main board is off, the light blinks. When the main board is working and the UPS is in the low power mode, i.e., the current is less than 50mA, the light also blinks. Otherwise, the light will be steady.

  With regard to the colors, LED1 shows the state of charge (SoC), i.e., the level of charge of the battery relative to its capacity, by colors. A blue light means charging of battery. A green light means the SoC is over 50%. An orange light means the SoC is less than 50% and the light turns red when the SoC is less than 15%.

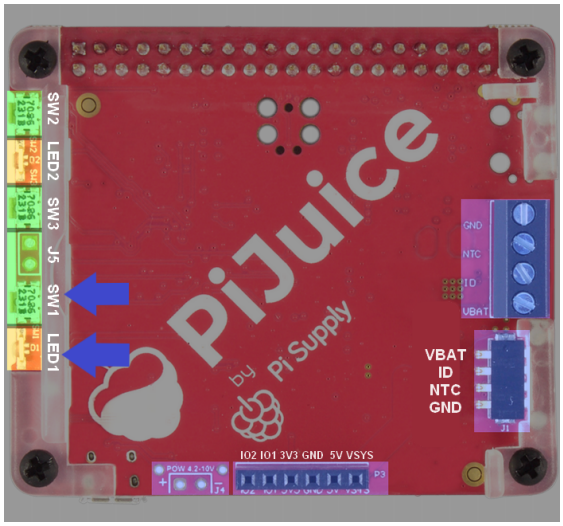

Figure 5: PiJuice HAT, UPS for Raspberry Pi.

- **SW1/J5.** To power on the computer, press the button shortly (release in 800 ms). To cut the power, i.e., turn off the computer forcibly, press the button for at least 20 seconds.

General instructions.

- **Turn on the computer.**
  To turn on the computer, press the SW1 button shortly, and the LED1 on the UPS will be a solid light when unplugged (if it has been charged properly).

- **Connect to the computer.**
  To connect to the computer, start a ssh connection with `ssh rescue@xxx.xxx.x.xxx`. The password is also rescue.

- **Charge the battery.**
  To power from the wall or charge the battery, plug the charger into the USB-C port on the main board.

- **Turn off the computer.**
  To turn off the computer, issue the command `sudo shutdown -h now` in the Raspberry Pi environment. If the computer is frozen, press the SW1 button for at least 20 seconds to cut off the power.

Appendix C Robot handling

This appendix describes instruction on how to handle the robot platform. Please read this appendix carefully and ask any question to the lab assistant if any arises.

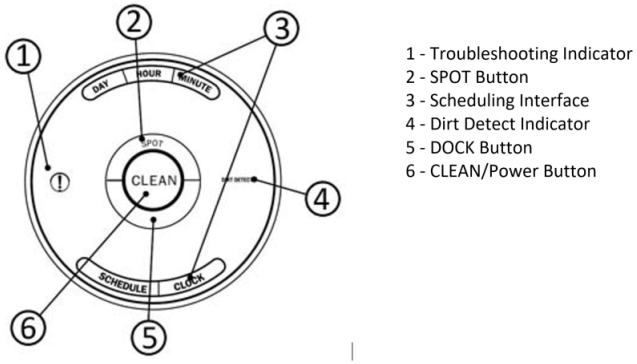

Figure 6: *iRobot Create* controls.

Robot usage.

Robot controls are shown in figure 6. There are some different buttons and LEDs, but during normal use, you will be using only CLEAN button (6), DOCK button (5) and Troubleshooting indicator (1). Every robot has a non dedicated Home base, where robot gets charged and standby.

- **Turn on the robot.**
  If the light around the CLEAN button is off, press the CLEAN button once and the robot will beep and the light will illuminate. Otherwise, the robot has been ready for action.
- **Pause and resume the robot.**
  The robot pauses when code execution is stopped and resumes when the codes are executed again. If the pause does not happen when the code execution is stopped, press the CLEAN button to pause. To resume the robot, press the CLEAN button again.
- **Charge the robot.**
  If the battery gets low before ending the searching, the robot returns to the Home base to recharge. At any time, the users can press the DOCK button and the robot will find its way back to the Home base. Alternatively, the users can move the robot directly to the Home base as well.
- **Turn off the robot.**
  To turn off the robot, press and hold the CLEAN button till the light around it is off.

- **Emergency power off.**
  If the robot goes out of control or the troubleshooting light (1) lights up, the users need to stop the robot in emergency situations. The first way is to issue a stop command from Raspberry Pi. The command is

```
echo -ne "\xAD" > /dev/ttyUSB1
```

  If the command doesn't work, you need to pull out the battery in the following steps: (1) Release the Raspberry Pi components from the robot using the velcro connections and the associated cables. (2) Turn the robot upside-down. (3) Remove the four screws circled in fig 7. The screwdrivers will be above the whiteboard in the Lab room. (4) Open the lid and pull out the battery carefully. (5) Finally, put the battery back and make sure the screws and the lid are secure. (6) Inform the lab assistant that you had to perform an emergency power off.

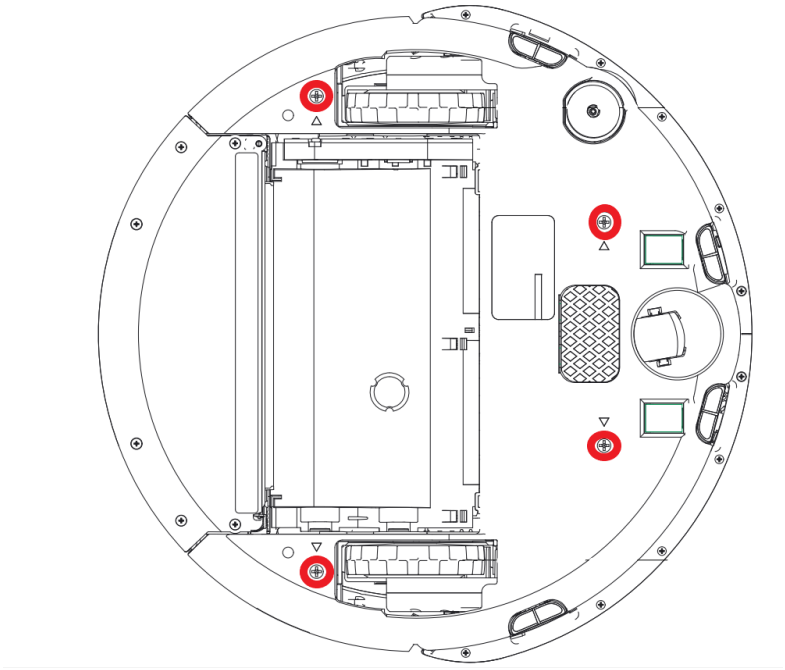

Figure 7: *iRobot Create* bottom

- **Move, carry the robot platform.**
  When carrying or moving the robot platform, grab it on the sides (left and right) of the robot. Do not hold on the front and back of the robot as you can damage the "bump"/avoid sensor. Do not use the handle on the robot as well.

Caution to be taken while using robot.

- **Be careful with robots while carrying or moving them!**
- **Be aware that the computer charger cable must be unplugged before running any code on the robot platform!**
- **If the robot is stuck, help it to get out as fast as possible so that internal electronics or mechanics are not damaged!**
- **If you see that your code does not run properly and it makes the robot go *"wild"*, stop execution of the code as described in *Pause and resume the robot*. If the robot doesn't stop, try the instructions described in *Emergency power off*.**
- **Ending work, turn off and charge the computers and the robots!**

Appendix D Getting started

This appendix provides detailed information that you need to start working with the labs.

One stationary computer and one robot platform is assigned to each student group (max 2 people in a group).
You will get a project folder that contains:

- source code of the robot agent
- mission control application
- scripts to setup the work environment and synchronise your program in the robot (computer)

In figure 8 a structure of the project folder is shown (items highlighted in magenta color are the ones you will be using).

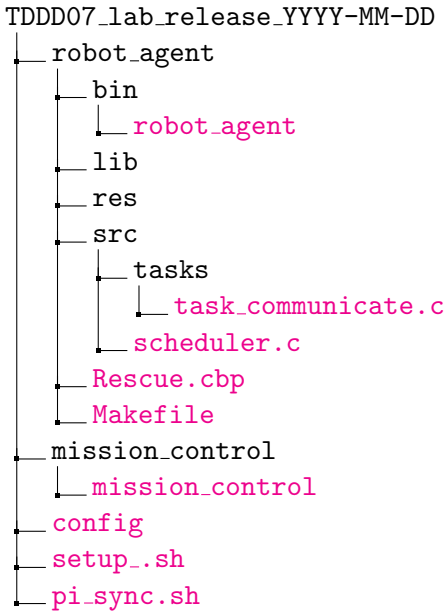

Figure 8: Project folder structure and important files (highlighted in magenta color).

Folder "robot_agent" contains a source code for the robot agent application. Under folder "bin" you will find the executable for the robot agent application. Folder "mission_control" contains an executable of mission control application. In main folder you will also find two bash scripts that must be configured before running. Script "setup_.sh" will generate proper configuration for the mission control application, robot agent application and will synchronise the robot (this script must be run once each time you start work in the lab). Script "pi_sync.sh" synchronises your compiled robot agent application on the robot.

Now lets go through few steps to prepare the robot and the project for first use. Some of the steps you will do only once and some you will need to do every time you come to the lab room or want to execute your code on the robot.

D.1 Prepare scripts.

(Do only once!)
You will receive configuration information from your lab assistant that must be copied into "config" file:

Open the "config" file with *gedit* or some other text editing software (*nano, emacs, etc.*) and copy your configuration into this file. Your "config" file should look similar to the one shown in listing 1.

Listing 1: Example of "config" file.

```
# Group (Number and Port)
LAB_GROUP=C1
LAB_GROUP_PORT=50001
# Robot (ID and IP)
ROBOT_ID=1
ROBOT_IP=192.168.1.101
```

When you have properly edited the configuration file, save and close.

D.2 Prepare robot platforms.

(Do every time you come to lab room.)
Now you need to prepare the robot platform (You will do this every time you come to the lab room). Follow the steps below:

1. Find the robot (or box where robot is placed) marked with your IDs.
2. Unplug the Raspberry Pi from its charger and move the robot away from its Home base.
3. Carefully move the robot to a place more accessible to you but still in the field (wooden area).
4. Make sure that 2 USB cables are connected to the Raspberry Pi (the one with the light on top).
5. Turn on *iRobot Create* by pressing the CLEAN button if the robot is not ready, i.e., the light around the CLEAN button is off.

The robot is now ready to run.

D.3 Run "setup_.sh" script.

(Do once every time you come to the lab!)
You will be using *Terminal* quite a lot, so get used to it and if you have not used it previously we suggest to read some tutorials online. Open *Terminal* and first run "setup_.sh". You may be asked to input a password (multiple times). Password is "**rescue**". If you do not see any errors and in the end you get message "Setup done!" then you can proceed, otherwise ask for help and show screen output to the lab assistant.

D.4 Run "pi_sync.sh" script.

"pi_sync.sh" script is run every time you want to copy your modified robot agent code files to the robot. The password for ssh connection is "**rescue**". This script also compiles the code files on the Raspberry Pi.

If necessary, you can connect via ssh to the computer, and navigate to "/TDDD07/LabGroupX/robot_agent", and run make on that folder, the compiler should run and compile your code files directly on the Raspberry Pi.

D.5 Run your application on the robot.

(Do every time you want to run your robot agent app.)

1. Open up *Terminal*.
2. Write "ssh rescue@xxx.xxx.xxx.xxx", where "xxx.xxx.xxx.xxx" is IP address of the robot and press Enter (in case the password is asked, it is the same as previously stated : **rescue**). It should connect to the robot.
3. Change directory to "/TDDD07/LabGroupX/robot_agent", where X is identification of your group/team. If you execute "ls" command, you will see that folder contains folder "bin" and executable "robot_agent" is under the "bin" folder. From this folder (robot_agent) you will run the robot agent application.
4. Now lets run "robot_agent" application. To run it you need to write " ./bin/robot_agent" and press Enter (it may ask for password).
5. At this point you should see "Hello World! Goodbye!".

If all steps went well without any problems you are ready to start programming your code and run it on the robot. Don't forget to have a look at the API on the website to know what the different functions are doing!

Appendix E Task pseudocode and timing information

This appendix contains simplified pseudocode for the seven robot tasks as well as some additional timing information that you will need to design your schedule. This presentation should be helpful to understand what is happening in the system but you may want to also look in the code. Especially, you should read to understand the dependencies among the tasks.

E.1 Mission task

Simplified pseudocode of the *mission* task is presented in Algorithm 1. Line 1 checks if the task is enabled to run. For the mission task, this will always be true. The other tasks in the robot are enabled or disabled when the mission task receives a "START" or "STOP" command from the Mission control. Lines 2-8 check if there is any pending mission data to be processed. Mission data can be either Victim data (lines 3-4), in which case the victim information is saved in a shared memory (list of found victim), or a command (lines 5-6) from the monitoring application which will then be executed. Finally, the task generates stream data which is put in a send queue to be processed by the communicate task (lines 9-10).

Timing information The refine task is only able to process information of a single tag from the RFID reader. If the RFID reader has read two or more tags since last polling the reader, only the oldest reading is treated by the refine task, the others will be kept in a buffer until the next time refine is run.

Algorithm 1 Pseudocode for the Mission task.

```
1: if mission_task.enable then
2:     for all data ∈ mission_queue do
3:         if data.type = VICTIM then
4:             insert data.victim into victim_list
5:         else if data.type = COMMAND then
6:             EXECUTE(data.cmd)
7:         end if
8:     end for
9:     data ← GENERATESTREAM()
10:    insert data into send_queue
11: end if
```

E.2 Navigate task

Simplified pseudocode of the *navigate* task is presented in Algorithm 2. In lines 2-4 the task checks if there is any pending pheromone map sectors shared by other robots and if there is, the local version of the map is updated. The robot also places pheromones on the local map based on its currently estimated position (line 5). After the map is updated it is put in a send queue (line 6). The send queue will be processed by the communication task ensuring that the map is broadcast to the other robots and the Mission control. Finally, based on the local pheromone map, the next move is calculated and saved in the shared memory (line 8).

Algorithm 2 Pseudocode for the Navigation task.

```
1: if navigate_task.enable then
2:     for all data ∈ navigation_queue do          ▷ Check incoming pheromones
3:         UpdateMap(map, data.pheromones)
4:     end for

5:     UpdateMap(map, robot_pose)                  ▷ Put robot pheromone on the map

6:     insert map into send_queue                  ▷ Make map available to be sent by
                                                   ▷ the communication task

7:     next_move ← PheromoneEvaluate(map, robot_pose)   ▷ Calculate next move
8: end if
```

E.3 Control task

Simplified pseudocode of the *control* task is presented in Algorithm 3. First, sensor data is requested from the robot platform, provided that a sufficient amount of time has passed since the last read (lines 2-7). Sensor measurements are incorporated in the localisation algorithm (particle filter) and a new pose of the robot is estimated (lines 3,4). The estimated pose is sent (via the communication task) to the Mission control so that the robot position can be visualised (line 5). Finally, the *next_move* motion command, given by the *navigation* task, is executed (lines 6-9).

Timing information The performance of the localisation heavily depends on the frequency at which the control task is run. If the control task is run very seldom, then the accuracy of the odometry readings will be reduced. Moreover, the control task should not be run too often since the interface to the robot does not cope with being polled at high frequency. Therefore, there is a timer to prevent the odometry data being read too often (not more often than every 300ms). If the task is run more often than every 300ms, it will only accumulate more localisation error.

Algorithm 3 Pseudocode for the Control task.

```
1: if control_task.enable then
2:     if read_timer > min_period then
3:         sensor_data ← ReadSensors()                          ▷ Read sensor data
4:         Reset(read_timer)
5:     else
6:         sensor_data ← ∅
7:     end if

8:     UpdateParticleFilter(sensor_data)      ▷ Incorporate odometer data in PF
9:     robot_pose ← ParticleFilterEstimate()                  ▷ Estimate robot pose

10:    insert robot_pose into send_queue

11:    if prev_move ≠ next_move then                              ▷ Check next move
12:        RobotMove(next_move)          ▷ Execute move if different from previous
13:        prev_move ← next_move
14:    end if
15: end if
```

E.4 Avoid task

Simplified pseudocode of the *avoid* task is presented in Algorithm 4. This task simply reads the bump sensor data and sends a turn command to the robot in case a collision is detected.

Timing information The avoid task should be run immediately *after* the control task, otherwise there is a risk that the control task overrides the avoidance command.

Algorithm 4 Pseudocode for the Avoid task.

```
1: if avoid_task.enable then
2:     if Collision(sensor_data) then                             ▷ Check bump sensor
3:         RobotTurn()                                  ▷ Turn the robot if collided
4:     end if
5: end if
```

E.5 Refine task

Simplified pseudocode of the *refine* task is presented in Algorithm 5. The task first checks the RFID reader (line 2). If a tag is read and it is is registered in the environment definition then the particle filter is updated and run based on the position of the tag (lines 3-6). If a tag has been read but is not registered then it is considered as a victim and its UID and location is sent to the *report* task (lines 7-11).

Algorithm 5 Pseudocode for the Refine task.

```
1:  if refine_task.enable then
2:      tag ← RfidRead()                                          ▷ Read RFID
3:      if tag ∈ LocalisationTags then     ▷ Check if tag is defined in the environment
4:          UpdateParticleFilter(tag)              ▷ Incorporate RFID tag data in PF
5:          ResampleParticleFilter()                              ▷ Resample particles
6:          robot_pose ← ParticleFilterEstimate()                 ▷ Estimate robot pose
7:      else if tag ≠ NULL then                    ▷ A non-localisation tag was found,
                                                   ▷ the tag must represent a victim
8:          victim.id ← tag                                       ▷ Save victim data
9:          victim.x ← robot_pose.x
10:         victim.y ← robot_pose.y
11:         victim.report ← true                                  ▷ Set "report" event
12:     end if
13: end if
```

E.6 Report task

Simplified pseudocode of the *report* task is presented in Algorithm 6. In line 2, the task checks if a victim is received from the *refine* task and in line 4 it checks if it already has been found and reported or not. In case the victim has not been reported yet, information about the victim sent to the *mission* task for registration in the victim list and put in the send queue to be disseminated via the communication task (lines 4-6).

Timing information The report task will only treat one victim at a time. If the refine task runs more often than the report task, and the refine task outputs two victims in a row, the report task will only report one of the victims (losing the other one!).

Algorithm 6 Pseudocode for the Report task.

```
1: if report_task.enable then
2:     if victim.report then                          ▷ Check if any possible victim found
3:         victim.report ← false                                  ▷ Reset "report" event
4:         if victim ∉ victim_list then                           ▷ This is a new victim
5:             insert victim into mission_queue       ▷ Inform mission about the victim
6:             insert victim into send_queue          ▷ Disseminate victim information
7:         end if
8:     end if
9: end if
```

E.7 Communicate task

Simplified pseudocode of the *communicate* task is presented in Algorithm 7. This task receives and transmits packets through the network. All the information from tasks that is supposed to be sent on the network, goes through the communicate task. The task takes all messages in the send queue, encode them as UDP broadcast packets and puts them in a queue at the MAC layer (lines 1-5). Packets received from the network are decoded into data structures and forwarded to proper tasks (lines 6-9).

Timing information In lab 2 your robot will only be allowed to communicate every 1000ms, which might be worth considering when implementing lab 1.

Algorithm 7 Pseudocode for the Communicate task.

```
1: while send_queue ≠ ∅ do                                  ▷ Send all queued messages
2:     msg ← RemoveFirst(send_queue)
3:     packet ← EncodeUDP(msg)
4:     Broadcast(packet)                    ▷ Puts the packet in a MAC-layer queue
5: end while

6: while ReceivePacket(packet) do                           ▷ Receive all buffered messages
7:     msg ← DecodeUDP(packet)
8:     HandleMessage(msg)                   ▷ Redirects incoming data to the proper tasks
9: end while
```

Appendix F Report requirements

A significant part of completing this lab series is to write a lab report. A single report will cover all labs.

The following general requirements should be met:

- The report should be well-written, well-structured and well-formatted. This means it should have an introduction and a conclusion, sections and subsections relevant for the content, full sentences, consistent formatting, and all figures and tables should be explained in the text.
- The file name of the report must be `TDDD07_lab_report_X_id1_id2.pdf`, replacing `X` with the name of your group (e.g., C5), and `id1` and `id2` with the LiU-IDs of the group members.

The following content should be covered in your report with regards to **lab 1**.

- Results from your execution time measurements of the tasks, together with an explanation of how you choose your final estimated WCET values.
- A table with the minimum and maximum possible periods for each task, as well as the value chosen by you for your schedule. You should justify each value with measurements or a logical argument.
- The schedule you have designed.
- An explanation of the tests you performed to evaluate the performance of the system for lab 1.
- The results of the evaluation of lab 1. This should include assessing the amount of deadline overruns of your tasks, how well you meet the application criteria detailed in Section 3.2, and measured accuracy of the location of the reported victims.

For **lab 2** you need to document the following points.

- Explain how you prioritise different message types and why you give such priorities.
- Describe the mechanism you use to determine which messages to send, and which messages to discard at each run of the communication task.
- Description of the tests you performed for lab 2.
- The results of the evaluation of lab 2. This should include assessing how often you receive the go-ahead message (and the causes of not receiving them) and the amount of stream data you are able to transmit in the time slot. You should also include how the performance metrics of lab 1 were affected by lab 2.

(Optional) If you have any comments, ideas, suggestions that would improve the labs, **please write them in the end of the report so that we have something concrete to change in the lab setup/compendium next year**.