EE2L21

Manual

EPO-4: Autonomous driving challenge

May 31, 2022

Delft University of Technology

Alle-Jan van der Veen
Jorge Martinez (editors)

TU Delft Delft University of Technology

Challenge the future

The Project autonomous driving challenge (EPO4) was conceived in 2011 as a semester project for second-year BSc-EE students. Since then many people have contributed to it:

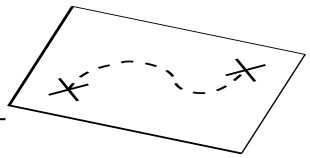
Paul Bauer
Gabriel Delgado Lopes
Gerard Janssen
Dimitri Jeltsema
Bert-Jan Kooij
Oleg Krasnov
Ioan E. Lager
Nick van der Meijs
Jelena Popovic
Alle-Jan van der Veen
Jorge Martinez
Carolina Varon
Alexander de Graaf
Fred van der Zwan

J.W. van der Merwe
Eric Roeling
Tim Velzeboer
Matthijs Weskin
Wessel Bruinsma
Reinier Prins
Conchita Martin
Joris Belier
Martijn Berkers
Teun de Smalen
Niels de Koeijer
Wouter Kayser
Richard Eveleens

Contents

Chapter 1

INTRODUCTION



Contents

Welcome to EE2L21 EPO4: Autonomous Driving Challenge. In this 5 EC project, you and your team will take a remotely operated electric toy car, and make it drive autonomously from A to B. The car, called "KITT" for nostalgic reasons, has a positioning system, and a communication system to a base station. It also can sense obstacles. And if all else fails, you can shout some voice commands to it as well. During a mid-term competition and a final competition you are asked to demonstrate your achievements in several "tasks" i.e., races of increasing difficulty.

Like in the previous EPO projects, the objective of EPO4 is to apply and integrate the EE knowledge you have acquired so far, in particular signal transforms, telecommunication, system modeling and control, and digital signal processing. We will also be using linear algebra and (Matlab) programming skills. Optionally, you can apply some machine learning algorithms as well.

This manual starts with a description of the overall project. As this is a complex task, it is split into a few modules which run sequentially or in parallel, each contributing to the overall goal. Also, you have already designed a TDOA distance estimation algorithm in the lab sessions of EE2T11 Telecommunications A, and you will now integrate this into the system.

1.1 SCOPE

The overall goal of the project is to take a standard toy car, with added functionality so that it can be remotely operated, can be located, can detect obstacles in front of it, and can communicate with a base station (PC or laptop running Matlab), which performs calculations such as location estimation, trajectory tracking and collision avoidance (see Figure 1.1). At start, the car is positioned at a known location. The

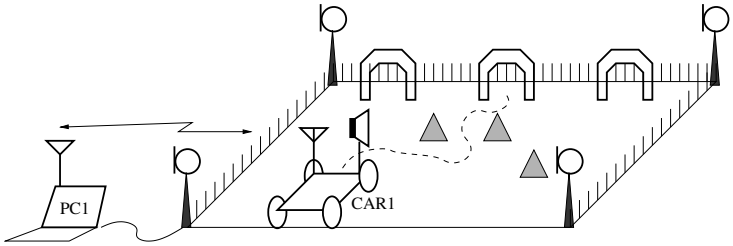


Figure 1.1: General set-up

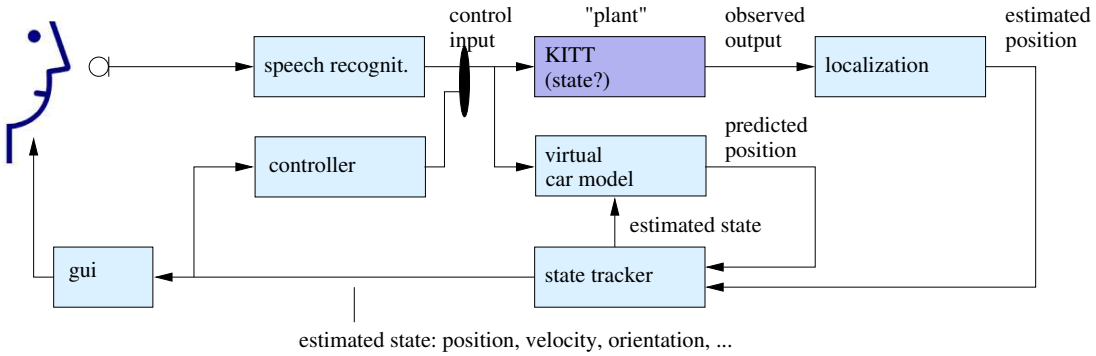


Figure 1.2: System overview

car should drive autonomously to the target, meanwhile avoiding obstacles such as boxes and competing cars. The car is equipped with an anti-collision sensor and a localization beacon. A bidirectional RF link will provide a connection of the car to a base station (PC) that allows to transfer the anti-collision ranging measurements, and to receive actuation instructions (motor-control and control of the equipment on the car).

Figure 1.2 shows the setup as a control system. Central is the "plant", i.e., the real car. You give it control inputs (drive and steering commands) and estimate its position from the audio beacon signals. Optionally, you can use a "digital twin" virtual Matlab model of the car, which should respond similar to the real car, so that you have an alternative way to estimate the position. From these position estimates, you can then track other state parameters such as velocity and orientation. This is input to your controller that computes the next control action, so that you arrive at the desired position B.

A graphical user interface (GUI) also plots the estimated state (location, orientation). If you see things go wrong, then with voice control, you can try to steer the car in the right direction or ensure an emergency brake.

This is a complex assignment, which is therefore split into a number of blocks or modules:

- (optional) Machine learning for speech recognition,
- Wireless link; anti-collision sensors.
- Car model, state/trajectory tracking and control,
- Localization using audio communication (using results from EE2T11 Telecommunications A practicum),
- System integration.

The speech recognition system is optional, as you might not have followed EE2ML1 Introduction to Machine Learning. Even so, Module 2 provides a 1-afternoon crash course in machine learning that should be sufficient to get you going.

The total duration of the project is about 8 weeks; each week consists of two sessions in the lab and two half-days of self study, preparations of the lab assignments, homework and report writing.

1.1.1 Objective

(Optionally) design and implement a speech recognition system. Integrate an anti-collision sensor, and design and integrate a localization and control system for an electric toy car, to drive autonomously and/or with limited voice overrides. The objective is to reach a target location accurately meanwhile avoiding possible obstacles. Document the designs in your reports which describe the design choices and evaluate the test results. Present and defend your reports and designs. Compete in "challenges"; races comprising tasks of different difficulties.

1.1.2 Design specifications

At the end of the project, you have built a system with the following specifications:

1. The *detection of obstacles* module consists of a pair of off-the-shelf ultrasonic transmitters/receivers that can detect objects in front of it.
    - The maximal range of the detection is about 5 m.
    - The system is mounted on the car.
    - The system is controlled by the basestation PC, and read out via the RF link.
    - A bidirectional RF communication link allows the basestation PC to control the car, and read out sensor data. This system is already provided.
    - The link is provided by off-the-shelf Bluetooth modules, which are controlled by a microcontroller on the car, and an interface on the basestation PC.
    - The microcontroller also generates motor control signals using basic commands coming from the PC.

2. During the *control and trajectory tracking*, the aim is to stop the car in front of an object (mid-term challenge) and to drive the car to a predefined location (final challenge).

    - (mid-term challenge) A basic control system allows the car to drive straight, and stop within time limits at a predetermined distance before the wall. Control input is provided from the distance sensors.
    - (final challenge) A control system (either autonomous, or with a "human in the loop") drives the car towards a target. Control input is from a localization algorithm that uses an audio beacon on the car.
    - Location information comes from a localization module; it will have a limited accuracy and provides location estimates only a few times per second. The orientation (pointing direction) of the car is not obtained from the localization module.
    - (optional) The state tracking is possibly enhanced by estimates from a virtual car model.
    - (optional) Using machine learning concepts, a speech recognition system is able to distinguish a restricted set of commands to specify commands such as "move forward", "stop", etc. (in your own language), which provide control inputs to the virtual car.
    - The control actions (car steering commands) are computed by a control module, with possible overrides by voice commands. A more advanced system also implements obstacle detection and avoidance.
    - "Open loop" solutions will not be accepted. These are solutions with hard-coded control actions that can only address a single scenario and do not use sensor inputs.

3. (final challenge) *Localization* is done by transmitting a repetitive beacon signal from the car. This is an audio signal with programmable parameters, and it may be corrupted by interfering signals emitted by other cars present in the field, and by surrounding noise.

    - The audio signals from all cars are captured by multiple microphones. The analog microphone signals are sampled and made available to Matlab.

      The received signals are not synchronized to the beacon (i.e., there is an unknown time offset) and the beacon transmits at most 10 pulse trains per second.
    - A map of the surroundings, the locations of the microphones, the starting point, the target, and possible waypoints is known. However, unexpected obstacles may be present.
    - The starting point and initial orientation (direction) of the car is known.
    - The estimation of the location is done in Matlab.
    - The location updates should be frequent enough and accurate enough for the trajectory tracking to function. Minimal numbers are at least one update per 2 seconds, with an accuracy of 5 cm or better.
    - It is permitted to stop the car to perform a location estimate.

4. During the mid-term and final challenge, the car is to drive autonomously and/or by voice control after a start command, controlled by the basestation PC.

- It is suggested that the programs on the PC are written in Matlab.
- The main microcontroller on the car is preprogrammed and has as tasks: (1) maintaining the Bluetooth link, (2) driving the servo motors for left/right and forward/backward, (3) reading out the battery voltage, (4) enable/disable the audio beacon and control its parameters, (5) reading out the obstacle detection system.
- During online operation, the PC (Matlab controller) has as tasks: (1) monitoring the battery voltage, (2) reading the sampled microphone data streams, (3) calculating the current location, (4) processing the obstacle detection data, (5) computing steering actions for trajectory tracking, (6) transmitting and receiving data over the RF link, (7) maintaining a status screen (dashboard), (8) optional: decoding voice input, (8) optional: adjusting the trajectory in case of a detected obstacle.

1.1.3 Competitions

Two demonstrations or "competitions" will be held. Consult the Brightspace page of the project for the exact dates when the competitions take place. The tests will be described briefly in this section, and in more detail in Chapters 6 and 9. The detailed rules of the competition will be made available shortly before the challenge.

The mid-term competition The mid-term competition is held after 2 weeks (4 lab sessions) and is a basic feedback control test. The objective is to drive forward and reach the wall as quickly as possible, at a predefined distance. For this competition, you drive only on a straight line (no steering). The ultrasonic sensors are used to measure the distance to the wall.

The design objectives are both speed and accuracy. The suggested feedback control system is based on rather straightforward "bang-bang" control principles, explained in Module 4.

This challenge consists of two cases: (1) start at a close distance (about 3 meters) which is in range of the distance sensors; the car will not reach full speed before it stops, and (2) start at a far distance (about 5–6 meters) which is out of range of the distance sensors; the car may reach a constant speed before it stops.

(Optional) If you implement a basic speech recognition system, three commands are recognized: "setup", "go" and "stop". The suggested speech recognition system is based on machine learning concepts and developed in Modules 2 and 3. If you choose this option, you only have to consider the "far distance" challenge.

The final competition The final competition starts at a known location and orientation. At a start signal, you have to drive autonomously to a target, optionally assisted by voice control.

If you complete the task successfully, you may try more difficult tasks including more waypoints, obstacles, and other cars. The winner is the team who completes the most tasks, or the fastest team if more teams complete all tasks. During the race it is not allowed to touch the car (or to use keyboard commands).

Speed is not the main requirement in this challenge. By driving faster, you may reach the target faster, but trajectory tracking becomes more difficult, location updates must be computed faster, and you have less time to avoid obstacles.

Important to note is that "open loop" solutions will not be accepted. These are solutions with hard-coded timings or control actions that can only address a single scenario and do not adapt to sensor input.

The requirements of the final challenge are completely different than those of the mid-term, although you will be able to re-purpose many system components.

1.1.4 Reports

The project outcome is documented in a mid-term report and a final report.

Reports follow a structured approach, which you have learned in previous projects. Use a to-the-point, concise but complete reporting style. Provide an appendix with all Matlab code.

The exact deadlines for handing in the reports are listed in Brightspace.

Submit your reports through the corresponding submission folders in Brightspace.

Mid-term report The mid-term report documents on the first modules (hardware aspects, machine learning, speech recognition, control system), and describes your design of a system to reach the spot. It contains your project design choices and problems you encountered until the mid-term challenge.

More detailed instructions are given in Chapter 6. Apart from the quality of the work (speed and accuracy of the results) and the quality of the report (systematic approach, motivation of design choices), you are judged regarding project skills such as planning and teamwork.

The mid-term reports are discussed during subsequent labdays.

Final report The final report documents on the design choices, the design itself, and the expected capabilities of the complete system. The focus is on your findings and measurement results and the corresponding conclusions. More detailed instructions are given in Chapter 9. Also here, you are judged regarding project skills such as planning and teamwork.

The final report is discussed during the presentation/defense session.

Note that the final report is quite independent of the mid-term report. Although some parts may be reused, the design objectives are very different, and so is the resulting system.

1.1.5 Presentation and discussion

In week 9, you present and defend your final report in front of an examination committee. The examinators will ask questions about your design choices and aspects of team work. This will be part of your grade.

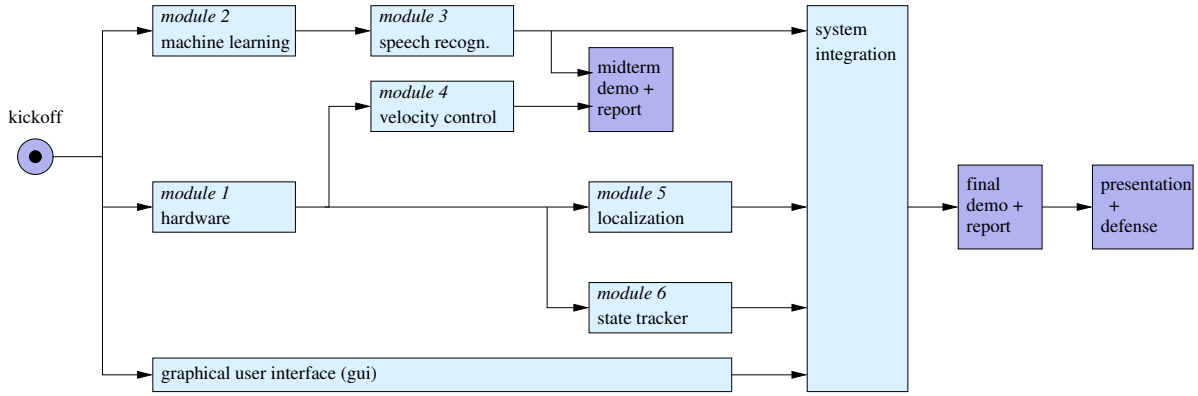


Figure 1.3: Project overview

1.1.6 Teamwork

You are asked to work in teams, using the processes as learned during previous EPO projects. Teams usually have a size of 4 students.

Two teams on different days share a car and associated hardware. Each week a team will have 2 sessions in the lab, and 2 half-days "unscheduled" time for homework, reading tutorials, collecting documentation, visiting consultants if needed, and report writing. The scheduling of the teams will be announced on Brightspace.

In your reports, show a work breakdown and time schedule (planning and actual realization). You will be judged on planning and team work.

A possible approach to organize team work follows the "Scrum" or "Agile" methodology. In this approach, a project evolves iteratively, where in each iteration ("sprint") a functional system is produced. A summary of Scrum is given in Appendix D.

1.2 PROJECT OVERVIEW

A schematic overview of the project is shown in Figure 1.3. The project consists of several tutorial modules and modules that provide system blocks.

1.2.1 Module 1: Communication and anti-collision sensors

After this module, you can drive the car using Matlab commands, and will know how the obstacle detection sensors work.

1.2.2 Module 2 (optional): Machine learning tutorial

After this module, you have touched the basics of machine learning. This will contain nothing new if you followed EE2ML1, except that Matlab commands are used instead of Python.

1.2.3 Module 3 (optional): Speech recognition system

After this module, you have implemented a basic speech recognizer, allowing you to decode a small number of voice commands.

1.2.4 Module 4: Velocity control

After this module, you have implemented a control system to drive on a straight line and reach a position in front of the wall as accurately as possible within a prescribed time.

1.2.5 Graphical user interface

The GUI is the heart of your system. It has a dashboard showing the arena, the status of your car, and some buttons for basic control. Its functionality evolves over time. This is where system integration happens.

1.2.6 Mid-term challenge

Each team is called separately to demonstrate the system so far: driving on a straight line to reach a spot within a set time. Optionally, you may use voice commands to start and stop the car. A mid-term report is submitted.

1.2.7 Module 5: Localization using audio communication

Starting point for audio localization is your Matlab code from EE2T11 Telecommunication A practicum to estimate the time difference of arrival of a pair of microphone signals. You extend this to estimate the location of the car by combining the TDOAs of all pairs of 4 or 5 microphones.

1.2.8 Module 6: State tracking and control algorithm

The location of the car is only estimated, and you do not know its complete state: its velocity and orientation are unknown. There are also annoying time offsets. You can devise simple or more complicated algorithms to estimate these. Once you know the state, your control algorithm decides where to go!

A virtual car model is provided that you may integrate in the design, or use to test your algorithms outside of the lab.

1.2.9 System integration and final challenge

This is where it all comes together. And where you find out the essence of system engineering: namely that it is essential to have fully tested and qualified subsystems, or else the overall system will surely fail.

A final report is written, and submitted the day after the final challenge.

As is apparent from the figure, several of the activities can run in parallel (although coordination is required). E.g., you could have separate team members work on speech recognition, localization, state tracking/control, and the GUI/system integration.

1.3 EDUCATIONAL OBJECTIVES

General learning objective: To integrate different technical areas related to electronic systems, signal processing and control. The educational objectives are:

- Increased skills in building and testing electronic systems. Software skills mainly consists of advanced Matlab, and a bit of Simulink.
- Increased skills in measurement techniques, e.g., wireless channel measurements and audio measurements.
- Application of course material from a variety of courses: control systems, linear algebra, signal transformations, digital signal processing, telecommunication, machine learning.
- System engineering skills: designing and testing sub-systems, and integrating these into a complete system.
- Increased academic skills related to project management: managing an open and complex assignment, planning, acquiring background literature, working in teams (distributing tasks among team members, communicating within and among subgroups), reporting, oral presentation.

1.4 FACILITIES

1.4.1 Bring your own device

While working remotely, you must have access to a laptop/PC running Matlab, and you will need a microphone if you work on the speech recognition task. In most cases, a laptop is more convenient than the fixed PC.

1.4.2 Lab support

The EPO4 labs are carried out at the Tellegen Hall facilities. Each week, a team will have 2 half-days access to the lab, and 2 half-days "scheduled preparation time" for reading tutorials, collecting documentation, finishing homework assignments, visiting consultants if needed, and report writing.

The following support is available:

- *Student assistants* are your primary help. Each assistant supports up to four teams. Assistants also check presence and progress.
- *Instructors* and practicum coordinators are available at each lab session. They will grade your reports.
- *Technical support* by student assistants and/or the technicians at the facilities.

1.5 TIME SCHEDULE AND DEADLINES

EPO4 has 14 scheduled lab mornings/afternoons, distributed over 8 weeks. Interleaved with these, there are unscheduled preparation/homework mornings. In week 8 there is a common morning for the final challenge, and in week 9 you will be called for a final presentation followed by a discussion. The total labtime is 60 hours, preparation/homework time is budgeted at 80 hours, for a total study load of 5 EC.

Visit the Brightspace page of the course for the detailed working schedule. It contains the dates of the challenges and the report deadlines. It also contains a suggested time planning regarding modules, such that you can meet the deadlines. You do not have to stick to the suggested planning, but specific staff members are available following the given planning, and the deadlines are firm.

1.6 GRADING

Your grade depends on the following:

- Mid-term report, combined with your performance during the mid-term challenge (30%);
- Performance during the final challenge (20%);
- Final report (35%);
- Oral presentation and defense (15%).

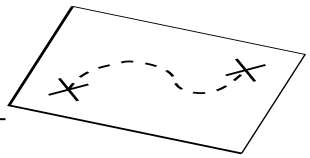
If you do not hand-in your reports in time, this will affect your grade.

Teamwork is important. Your individual grade may differ depending on staff observations and a peer review system.

If your final grade is insufficient, you may have a chance to improve your grade by improving your mid-term report or final report.

Chapter **2**

MODULE 1: HARDWARE



Contents

KITT is a remotely operated vehicle that has the ability to detect obstacles by "sensing" its environment. The remote operation requires a wireless communication link. The obstacle detection requires sensors that can remotely sound the environment and send the retrieved information to the processing unit (the computer) in predefined packages. This is a simple form of radar.

In this chapter, you familiarize yourself with the project, your team, and the hardware. Get to know your partners, their strengths and weaknesses. With the team, you make a planning, define roles, and distribute tasks. Let's get KITT ready to race!

Learning objectives The following is learned and practiced in this assignment:

1. Getting to know your team and setting up a collaborating environment
2. Setting up the Bluetooth connection to the car
3. Driving KITT using the keyboard
4. Requesting data from KITT and retrieving the information
5. Start on a GUI
6. (optional) Inspect the virtual model of KITT
7. Understanding the accuracy and the limits of acoustic distance measurements

Deliverable As part of the mid-term report:

A section describing how you function as a team (your planning, and the reality), how you distribute the work and manage deadlines.

A section describing the hardware, your experiments (regarding communication and sensing) and *an analysis of this*: what are the key problems to address, given the limitations of the hardware?

What are sources of error, and can you calibrate for this? Think e.g., of unknown communication/measurement delays, a slow response of the cars, imperfect speech recognition. Hint: for the mid-term, an analysis of the various delays in the system is essential!

A section describing the GUI. This becomes part of a section on system integration.

Preparation Read this chapter and the KITT datasheet (download it from Brightspace). Also refer to Appendix D on project management using Scrum.

Prepare a measurement plan. Some considerations can be: what do you want to know, what are you going to measure, where are you going to measure, who is going to measure, which calculations do you need.

Time duration One lab session for the full team, plus one lab session for half the team (running in parallel with Module 2, if you choose the machine learning option). One homework session for reading and preparing the report.

What is needed

- KITT
- Laptop running Matlab; and/or the Tellegenhall computer
- Brightspace:
  - Datasheet for KITT and that of the ultrasonic sensors `SRF02`.
  - Matlab executable file `EPOCommunications.mexw64`. Place it in a directory where Matlab can find it.
  - (optional) Zipfile `carmodel.zip` with a virtual KITT implementation. Unzip the file and install the files in a directory where Matlab can find them.

2.1 GETTING TO KNOW YOUR TEAM

To form a team with new people is not always easy. Let's start with a simple task: filling out a document together. Please look on Brightspace for the document `epo4_kickoff.docx`. The document asks questions about how you want to collaborate as a team. Submit the document in your Brightspace folder.

Also refer to Appendix D, which describes Scrum, a technique for (ICT) project management. You should find ways to brainstorm as a team, find possible approaches to a problem (analyze the problem first!), and keep everyone busy, involved and synchronized.

The main thing to decide is if you want to dedicate two team members on the machine learning task (voice recognition). If you do, then the requirements for the mid-term are slightly different. Browse over Chapter 6 to know what these requirements are.

Figure 2.1: Appearance and dimensions of KITT (before modifications)

2.2 GETTING TO KNOW KITT

At the start of a project, it is important to familiarize yourself with the hardware, in this case the car. In this module, we collect the main information that might be useful. Perhaps most relevant is the list of commands that are sent over Bluetooth to control the car.

2.2.1 Hardware

KITT is based on a repurposed car model, the Traxxas E-MAXX. Figure 2.1 shows a picture. Other specs are shown on the company website. Initially the car was equipped with supercaps that could be contactlessly charged from coils at the bottom, but the current version uses battery packs for safety reasons.

The hardware aspects of KITT are described in a datasheet that you can download from Brightspace. It gives an overview of the various components, such as the motor controller, LCD status indicator, ultrasonic sensors, and most importantly, the communication module.

Many of the components are combined on a single custom made MCU board installed on KITT (see Fig. 2.2). The core of this board is formed by the microcontroller, an NXP LPC4357 chip with an ARM Cortex™-M4/M0 core. The MCU board further contains the Bluetooth module, connectors for all peripherals and an amplifier for the audio beacon (used for localization in Module 3). In Figure 2.3 a schematic overview of the connection is shown.

The MCU is fed from the rechargeable batteries using a buck converter that converts the voltage to 5V DC. Using an LM1117-3.3 also 3.3V is generated.

The Bluetooth communication with KITT consists of the following elements:

- On car: Roving Networks RN-41 I/RM, Onboard Bluetooth module with UART control.
- On PC: LM Technologies LM506, USB Bluetooth v4.0 dongle with Broadcom BCM20702 chipset.

Figure 2.2: The controller board in KITT

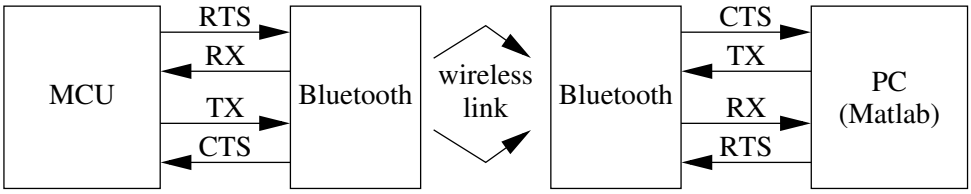


Figure 2.3: Overview of the communication between KITT and the PC (Matlab)

(a) On car (b) On PC

Figure 2.4: The two communicating parts of the Bluetooth link

(a) Leds and status display (b) Power connector and motor switch

Figure 2.5: Status display, leds and power connector on the car

The Bluetooth module and USB Bluetooth dongle are *pre-configured*. The USB Bluetooth dongle must be connected to the host computer on which the controlling software runs. The relevant drivers are installed on the lab computers. If you want to run your controller on your own laptop, you can either use a built-in Bluetooth adapter or download the necessary drivers from the internet.

Figure 2.5a shows the status display and status leds on the car. The meaning of the leds is as follows:

- Red (2 times): 5 V and 20 V supply voltage present
- Green: (blinking) Bluetooth searches for connection, (steady) Bluetooth connected
- Yellow: (blinking) Bluetooth data transfer

Next to the leds is also small reset switch which will reset the MCU.

Figure 2.5b shows the side of the car. The green battery power plug should be connected either in the left or the right port (they are equivalent, the other port is used for contactless charging). The middle port is connected to a large switch, which is for switching on/off the motor controller. This allows to test the communication link without actually running the car.

2.2.2 Bluetooth link commands

As mentioned, the communication between the Bluetooth wireless interface and the MCU is pre-programmed. Your task is to write software to remotely control the vehicle by sending the relevant commands and reading the remote status and sensor information. Your first assignment is to make the scripts to connect to KITT.

The communication via the Bluetooth wireless interface is achieved via a serial port. The USB Bluetooth adapter must be connected to one of your computer's USB ports. The KITT datasheet provides details of the various commands and status strings that can be communicated over the Bluetooth link.

For your convenience, a Matlab function`EPOCommunications` is provided on Brightspace to help with transmitting and receiving data. This is precompiled C-code that should be placed in a directory where Matlab can find it. Here is how to use the `EPOCommunications` function; you can also type `EPOCommunications('help')` for a brief summary.

Opening the Bluetooth link Generally, at the beginning of a session a serial communication port to the Bluetooth adapter should be opened. A typical command is:

```
comport = '\\.\COM24'; % the actual COM port
                       % to use varies.
result = EPOCommunications('open',comport);
                       % open connection.
```

Verify the status of the connection!

If `result == 0` then an error occurred, typically the port is already open (close it first).

Requesting the car status The current status of the car is requested using

```
EPOCommunications('transmit','S');
                       % request status string
```

The status string reports the current drive commands, the ultrasonic sensor distance (cm), the battery voltage (mV), and the audio status (on/off) and control parameters (code word, carrier-frequency, bit-frequency, repetition frequency). The output structure of this command is:

```
*************************
* Audio Beacon: on
* c: 0x00000000
* f_c: xxxxx
* f_b: xxxxx
* c_r: xxx
*************************
```

```
* PWM:
* Dir. xxx
* Mot. xxx
*************************
* Sensors:
* Dist. L xxx R xxx
* V_batt xx.x V
*************************
```

A sensor distance of `999` means overload (i.e., out of range).

Driving the car The car is told to drive in a certain direction using two commands

```
EPOCommunications('transmit','D150');
                    % direction command
EPOCommunications('transmit','M150');
                    % motor speed command
```

More generally,

```
d = int2str(direction);
s = int2str(speed);
signal = ['D',d];
EPOCommunications('transmit',signal);
signal = ['M',s];
EPOCommunications('transmit',signal);
```

The first command sets the direction. A value of `150` means neutral. The minimum value is `100` and the maximum is `200`. The second command sets the speed. A speed of `150` means stop. Between `151-165` means to drive forward, and between `135-149` means to drive backwards. Unfortunately, there is a dead zone; the car may not start to move before `153` or so. The speed and deadzone also depend on the battery status. The command returns the current status.

Handling of the audio beacon The audio beacon is switched on or off using

```
EPOCommunications('transmit', 'A0');
                    % switch off audio beacon
EPOCommunications('transmit', 'A1');
                    % switch on audio beacon
```

The beacon signal is similar to what was used in EE2T11 Telecommunication A practicum, except that now it is possible to use an arbitrary carrier frequency, bit frequency, and repetition count. The code

length is 32 bits. The code itself is arbitrary. The following four commands are available to configure the behavior of the beacon (using example values):

```
EPOCommunications('transmit', 'B5000');
                    % set the bit frequency
EPOCommunications('transmit', 'F15000');
                    % set the carrier frequency
EPOCommunications('transmit', 'R2500');
                    % set the repetition count
EPOCommunications('transmit', 'C0xaa55aa55');
                    % set the audio code
```

The bit frequency determines the bandwidth used by the signal; the default value is 5 kHz.

The repetition frequency is specified by setting the repetition count. The relation between these two units is:

$$repetition\_count = bit\_frequency / repetition\_frequency$$

The repetition count is the number of bits the beacon waits before transmitting the code again. The minimum value is 32 (otherwise a new code is transmitted before the previous one finished). With the default value of the bit frequency (5 kHz) the shown (default) value of 2500 for the repetition count relates to a repetition frequency of 2 Hz (rather slow).

The fourth command is used to set the 32 bits code pattern. It is required to specify the code in C hex format (here as example `0xaa55aa55`; probably not a very good code). The default code is `0x00000000` which means no code is sent.

Closing the Bluetooth link At the end of a session, the communication port should be closed:

```
status = EPOCommunications('close');
                    % close connection
```

Here, `status` returns `'0'` if an error occurred, and `'1'` if the port is successfully closed.

Actually, it is a good idea to close the port also at the beginning of a session, just in case the communication port was left open by somebody else. **The Bluetooth connection is easily disturbed by leaving comm ports open, quitting Matlab, or removing the Bluetooth dongle without closing ports first.** Typically, in these cases Matlab refuses to open a serial port because some other process seems to own the port. You will lose lots of valuable time trying to resolve this, so pay attention.

Important **If you issue the Matlab command `clear all` you lose the pointer to the serial port, meaning you cannot close it but also cannot reach it.** You also cannot open a new connection since the port is still occupied. The only option is to restart the computer and set-up everything again. You will lose lots of valuable time!

2.3 ASSIGNMENT

Task 1: Test the Bluetooth connection Make scripts to open and close the serial connection. Request the status, send driving and steering commands. Take KITT for a spin!

Task 2: Implement a Matlab interface Write a script in Matlab to communicate with KITT. You must drive the car remotely and you must read incoming data. Structure your files in such way that you can reuse them later. Your script should be capable to:

- Drive the car properly (using keyboard commands)
- Read incoming data (i.e. measurement results, errors, etc.)
- Handle errors in the right way (e.g. emergency stop, resend commands, etc.)
- Visualize the commands and measurement results to compare the wanted values with the actual values (e.g. a dashboard).

Task 3: Start on a GUI The GUI is a dashboard by which you can easily control the car and observe its status.

You can use the (old) Matlab GUI Design Environment (`guide`) or (newer) `appdesigner` tool to set up a basic loop that scans the keyboard for events and updates a user interface, e.g., a dashboard with plot windows and status indicators. (You can also build something like this yourself with basic commands such as `uifigure`, `uicontrol`, or use other tools; an example is seen in the file `test_epo.m` found in `carmodel.zip`.)

Task 4: (optional) Inspect the virtual model of KITT During Corona times, a virtual model of KITT was developed which implements all its functionality, using the same interface as the real car. Its description is in Appendix A.

You may not need this model right now (or not at all, depending on your design), but have a quick look so you know it exists. This model has not been calibrated, so if you use it, you probably want to tweak the parameters to match its behavior to the real KITT.

Warning Although KITT's speed is internally limited, it can drive very fast. Equip your Matlab interface with an emergency stop interrupt when you press a button (e.g. `Esc`) to prevent damage to the car and surrounding objects.

2.4 DISTANCE SENSORS

We will now look in more detail at the distance sensors: they are essential for the mid-term challenge. As described briefly in the KITT datasheet, the car has two ultrasonic sensors, left and right, each consisting

Figure 2.6: SRF02 ultrasonic distance sensor

of a SRF02 module (see Figure 2.6). The data sheet of this module is also on Brightspace. It works by transmitting a pulse train at 40 kHz, and listening to its echo. The time until the first echo is received is measured and converted to a distance (cm). According to the SRF02 datasheet, the cycle period (time between two observations) has to be at least 66 ms. The modules are connected to the MCU and the interface is *pre-programmed*. The cycle time is fixed at 70 ms and in this period, the left and right sensor are started one after the other.

To have a good understanding of the limitations of the systems, you have to determine the working of the ultrasonic sensors.

2.4.1 Limitations of the system

The practical realisation of the SRF02 modules is simple and adequate for its purpose (parking sensors). However, the accuracy of the estimated distance is affected by many factors, such as: the mounting of the sensors in combination with the type of beams they generate, and the environment. Try to imagine how such phenomena and/or circumstances impact the ranging accuracy.

Moreover, the distance measured by the sensors must be relayed to the control system on the computer. The various communication delays are at the origin of additional errors, e.g., the car has already moved some distance by the time the measurement arrives at the computer. Make a rough estimate of the delays[1] and try to figure out how they, individually and/or in an aggregate manner, impact the performance of the complete control chain.

The goal of this exercise is to (1) to determine the accuracy and variance of the ranging information provided by the sensors, (2) to know if outliers can be expected and, if so, how to deal with them, and (3) think of calibration methods that limit or even eliminate the effect of these possible errors on the control chain. E.g., the data sheet does not mention the impact of interference of two (or more) sensors on each other.

2.4.2 Tasks

Task 1: Limitations of the system Discuss the limitations of the system along the lines mentioned above.

Regarding delays: how “old” do you think a distance measurement is when it arrives at Matlab? What is

[1] Hint: make a functional scheme of the complete chain.

the variance of that measurement? Where is the main bottleneck? (This is important information for the mid-term.)

What is the maximal range you can reliably measure distance?

Task 2: Static measurement of the sensor Perform measurements with the vehicle at stand-still. Introduce various obstacle configurations.

Also check the pointing direction of the sensors. If the car brakes strongly, its nose dives down and the sensors see the floor—does this result in erroneous readings? Are there any solutions to this?

Task 3: Dynamic measurement of the sensor Perform measurements with the vehicle in motion. Introduce various obstacle configurations.

Make a plot of the results. Plot the left and right sensor values. You will probably see a "staircase". Explain what you see in the plot.

Task 4: Delay measurements Measure the communication delays: how much time elapses between sending a drive command and receiving its status (including distance measurement)? This determines the maximum command refresh time, an important parameter for your control loop.

Make several observations and look at the resulting delays: are these consistent or are there outliers? Make a histogram of the delay measurements.

(This is important information for the mid-term. Try various possibilities, e.g. the delay in sending a drive command, and the delay in requesting a status.)

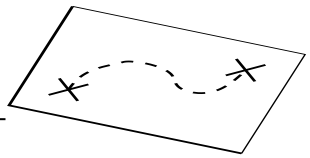
Task 5: Measurement data analysis and interpretation Make an analysis of the measured data with an eye on the possibility to compensate for the possible errors. Implement some strategies to calibrate your KITT. Include this data in your report.

Task 6: Datasheet In the datasheet of KITT (see Brightspace) some fields are left open. Fill in these values according to your measurements.

Include your measurements and document all tasks results in your report.

Chapter ***3***

MODULE 2: MACHINE LEARNING



Contents

This year, KITT becomes voice-controlled! We pretend that you can talk to the car via your microphone and give it instructions (next to any autonomous driving options that you may implement later after the Midterm). The voice control system uses machine learning and signal processing algorithms.

In this Module, we look at the basics of Machine Learning. In the next Module, we will apply that to design a basic speech recognition system. Some of the functions used in this Module are only available on Matlab R2019b and R2020a. Some problems can be encountered when using older versions.

If you took EE2ML1, you will not learn much new in this Module, except for the Matlab commands that may be used (rather than the Python commands you saw in EE2ML1).

Learning objectives The following is learned and practiced in this assignment:

1. Designing your own machine learning algorithm
2. Discriminating between sounds using machine learning
3. Selecting the most relevant features from a dataset
4. Evaluating the performance of a machine learning classifier

Deliverable As part of the mid-term report:

– A section describing your experiments (solutions of both assignments), in which you explain your feature selection approach, design of the classifier, the performance, and conclusions based on this. What are the main problems when discriminating between words and vowels, and how can you overcome these problems?

- Models designed for each assignment. You are expected to use the classifier learner app of Matlab. You do not need to write your own codes for the classifier, you can export these models from the app.
- Codes for both assignments

Preparation

- Read this module
- Read the introduction chapter of the book "Machine Learning Refined" by Jeremy Watt, Reza Borhani, and Aggelos K. Katsaggelos.
- Follow the different online lectures/videos indicated in this module. Some of the videos are introductory and contain information that might overlap with the book chapter. However, they offer different examples that help illustrate the topics.

Time duration One lab session and two home sessions.

What is needed

- Laptop/PC running Matlab R2019b or R2020a;
- Introduction chapter of the book "Machine Learning Refined" by Jeremy Watt, Reza Borhani, and Aggelos K. Katsaggelos. You can take this from Brightspace.
- Brightspace:
  - Matlab function `lda.m`
  - Dataset for word and vowel classification – `Data_UCI_EPO4.mat`.

3.1 ARTIFICIAL INTELLIGENCE vs. MACHINE LEARNING vs. DEEP LEARNING

Nowadays, the terms "Artificial Intelligence" (AI), "Machine Learning", and "Deep Learning" are used interchangeably despite the fact that they are completely different concepts, which are, however, related to each other (see Figure 3.1). AI is the umbrella term that deals with the development of intelligent systems capable of thinking and performing tasks like humans. Machine learning, on the other hand, is a sub-field of AI, that:

gives computers the ability to learn without being explicitly programmed.

Arthur Samuel (1959)

More formal definitions of machine learning are given by Kevin P. Murphy, in his book "Machine Learning, a probabilistic perspective":

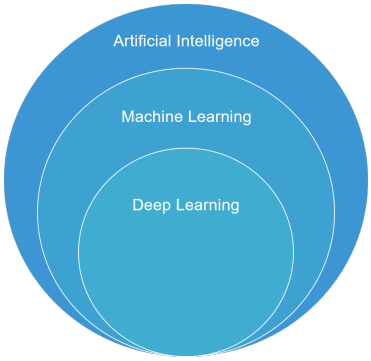


Figure 3.1: Relationship between artificial intelligence, machine learning, and deep learning

*Machine learning is a set of methods that can automatically **detect patterns** in data, and then use the uncovered patterns to **predict** future data, or to perform other kinds of **decision making** under uncertainty,*

and by Tom M. Mitchell in his book "Machine learning":

*Given a task T, a performance criterion C, and experience E, a system **learns from** E if it becomes better at solving task T, as measured by criterion C, by exploiting the information in E.*

Finally, deep learning is a subset of machine learning methods inspired by the way the human brain works. Deep learning algorithms are based on neural networks, are suitable for handling large amounts of data, and have proven to be superior to many traditional machine learning techniques in terms of performance. One of the crucial advantages of deep learning over traditional machine learning techniques is that no **feature engineering** is required. This is due to the ability of the different layers of neurons to learn and extract relevant and typically unknown information from the raw data. Often, feature engineering is a time consuming procedure that requires prior (i.e., expert) knowledge of the problem. At the same time, it adds intepretability to the algorithms, since it is possible to keep track of the different characterizations of the data. This will become clear further in this module. In deep learning, however, this interpretability is generally lost due to the fact that the user cannot keep track of the output of the many layers and neurons in the network. Apart from this, there are other advantages and disadvantages that must be considered when developing a machine learning algorithm: computational resources; amount of data available; and more important, the problem that needs to be solved.

For example, if the problem is to predict the arrival of a storm at your place, you may want to use the best algorithm that can alarm the people as fast as possible. At this point, you normally do not care if it was the humidity or the air density what deviated the storm towards your place. Therefore, here you may want to use a deep learning algorithm, where you sacrifice interpretability for performance. On the other hand, if what you want to do is to understand a disease, you may want to go for a feature engineering approach, where you know exactly what triggered the occurrence of the disease. For instance, was it the blood pressure? or the heart rate?

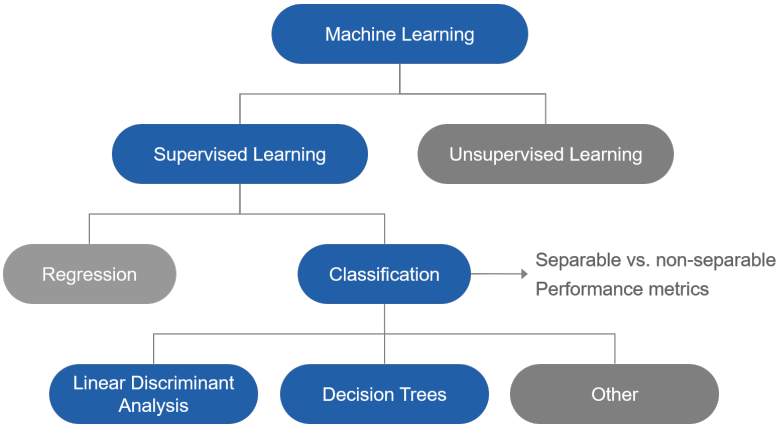


Figure 3.2: General overview of the topics covered in this module. The focus will be on the topics highlighted in blue.

In order to learn more about the differences between these three concepts, their advantages and disadvantages, watch the following online material:

- A 15 minute video, by edureka!, of the historical and general differences between AI, machine learning, and deep learning, that can be found here. This video also covers the advantages and disadvantages of deep learning. Important to get a general view on the topic!
- A longer video of 28 minutes, by Simplilearn, with more in-depth examples and a comparison between these concepts, can be found here.

In this module, the focus will be on traditional machine learning techniques, where feature engineering is required.

3.2 MACHINE LEARNING

For a general introduction to machine learning, watch Lecture 1.1 of the course given by Andrew Ng, co-founder of Coursera and deeplearning.ai. In addition, read the Introduction chapter of the book "Machine Learning Refined" by Jeremy Watt, Reza Borhani, and Aggelos K. Katsaggelos. It gives an overview of the different topics that will be covered in this section, which are highlighted in Figure 3.2.

There are different types of machine learning paradigms, including supervised and unsupervised learning, reinforcement learning and recommender systems. For EPO4, we will only deal with supervised problems.

The following online material, developed by Andrew Ng for Coursera, describe the main two learning paradigms:

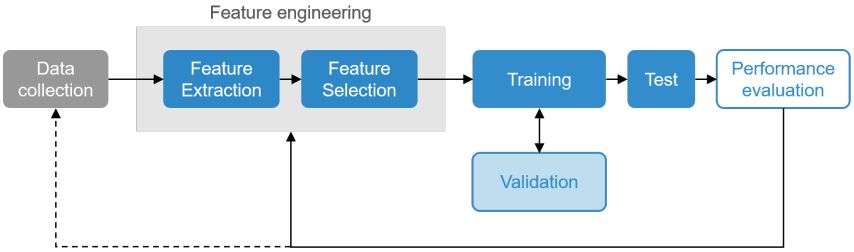


Figure 3.3: General pipeline to solve machine learning problems.

- Lecture 1.2 - Supervised Learning.
- Lecture 1.3 - Unsupervised Learning.

The main goal in supervised learning is to find a predictive function $f : \mathcal{X} \rightarrow \mathcal{Y}$, that maps instances contained in an input space $\mathcal{X}$ to some output space $\mathcal{Y}$. In other words, the goal is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that can predict an attribute $y_i \in \mathcal{Y}$ from input samples $\mathbf{x}_i \in \mathcal{X}$.

There are two main tasks in supervised learning: Regression and Classification. When y_i is a categorical or nominal variable (e.g., cat or dog; -1 or 1), the problem is called classification. Instead, if y_i is a continuous real-valued variable, the problem is called regression. Here we will only work with classification problems.

In order to illustrate this concept, we will solve the following classification problem. Let's assume that we want to distinguish the voice of a child from the voice of an adult. This problem can be solved following the general pipeline of machine learning, depicted in Figure 3.3. This pipeline is in fact used to solve any machine learning problem, with the exception of some deep learning structures, where no feature engineering is required.

3.2.1 Data collection

During this phase, it is crucial to collect as many samples as possible so that the machine learning algorithm can learn from a diverse set, where all the possible variants of the population are included. For this particular problem, samples refer to subjects $\{\mathbf{s}_i\}_{i=1}^{N}$, with $N = 200$, where we assume that 100 children and 100 adults were sampled from the general population. These subjects will, in theory, describe the underlying distribution of all children and all adults the machine learning algorithm will "know", or will be able to identify. This phase of data collection will be exercised in module 2. For now, we will assume that the data set we collected, which we will call the *training set*, denoted $\mathcal{D} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$, with $N = 200$, contains all possible examples we want our algorithm to learn from.

During the data collection phase, we assume that speech signals were collected from each subject, while pronouncing different vowels.

Probably one of the most important steps when designing a machine learning algorithm is the definition of the *training* and *test* sets. It is crucial to guarantee that the training set contains all possible examples

since this is the set that will be used to train the algorithms. Those algorithms will be evaluated using the test set, which should be completely independent. In other words, during training, no sample from the test set should be presented to the algorithm. The reason for this is to guarantee a good *generalization* of the algorithm. This means that the algorithm is able to perform the task on the test set, after learning its solution on the training set, with comparable accuracy in both sets.

3.2.2 Feature extraction

Now, we need to characterize each subject or each recording, using parameters or so-called **features**. This phase is called *feature extraction* and it requires certain levels of expertise so that the features can be useful to identify when a child or an adult is speaking.

From experience we know that the fundamental frequency, denoted F_0, of children's voice is higher than for adults. For instance, average values between 130 Hz and 220 Hz have been observed for adult males and females, respectively. For children, on the other hand, average values larger than 260 Hz have been reported for ages below 5 years. Apart from F_0, the formant frequencies F_n in children are much higher than for adults, typically higher than 4 kHz, while for adults they are often in the range 0.3-3.2 kHz. These differences become less evident while growing up.

Using this expert knowledge, we can characterize each subject using 2 features, the fundamental frequency and one formant frequency. Therefore, $\mathbf{s}_i = (\mathbf{x}_i, y_i)$, with $\mathbf{x}_i = (F_0, F_n)$ and

$$y_i = \begin{cases} 1 & \text{if } \mathbf{s}_i \text{ is child} \\ -1 & \text{otherwise.} \end{cases} \tag{3.1}$$

You can generate an artificial dataset using this expert knowledge, by means of the following code.

```
N_c = 100; % Number of children in the dataset
F0_children = 260 + 20*randn(N_c,1); % Fundamental Frequency
Fn_children = 4200 + 100*randn(N_c,1);% Formant frequency

N_a = 100; % Number of adults in the dataset
F0_adults = 130 + 20*randn(N_a,1);
Fn_adults = 2500 + 100*randn(N_a,1);

X = [F0_children Fn_children;F0_adults Fn_adults]; % Feature matrix
Y = [ones(N_c,1);ones(N_a,1)*-1]; % Class labels

scatter(X(Y==1,1),X(Y==1,2),'diamond')
hold on,grid on, box on
scatter(X(Y==-1,1),X(Y==-1,2))
xlabel('$F_0$ (Hz)','Interpreter','latex')
ylabel('$F_n$ (Hz)','Interpreter','latex')
```

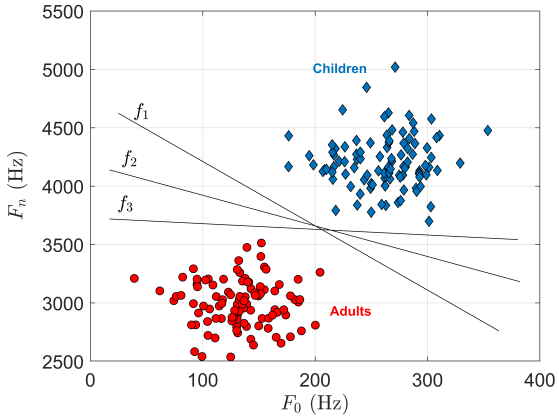


Figure 3.4: Artificial dataset of voice properties in children and adults.

Here, you created a dataset of 200 samples, with 100 samples drawn from the population of children, and 100 from the adult population. The input space consists of 2 dimensions, defined by F_0 and F_n, where both features were generated so that they follow a normal distribution for each class. The output space consists of two categories, making this a *binary classification* problem. The dataset is depicted in Figure 3.4.

This dataset belongs to the category of *separable* problems, since the classes can be clearly separated by a linear *decision boundary*. In this case, this boundary could be any function. Some candidates are the linear functions f_1, f_2, and f_3, indicated in Figure 3.4. Before finding this "optimal" function, it is important to select the features that are more informative and remove features that are irrelevant for the problem. This phase is called *feature selection*.

Let's assume that the dataset also contains information about the volume of the voice of the subjects, which for this simulation, varies around 75 dB. As can be expected, in normal circumstances, the volume of the voice is not a feature that differentiates between children and adults. Therefore, it does not contain relevant information to solve this problem, and we can simulate it using the following code:

```
V = 75 + 5*randn(N_c+N_a,1); % New feature
X = [X V]; % 3 features combined

scatter3(X(Y==1,1),X(Y==1,2),X(Y==1,3),'diamond')
hold on
scatter3(X(Y==-1,1),X(Y==-1,2),X(Y==1,3))
xlabel('$F_0$ (Hz)','Interpreter','latex')
ylabel('$F_n$ (Hz)','Interpreter','latex')
zlabel('$V$ (dB)','Interpreter','latex')
```

Figure 3.5 shows the dataset in a 3-dimensional space, where it is clear that the new feature does not

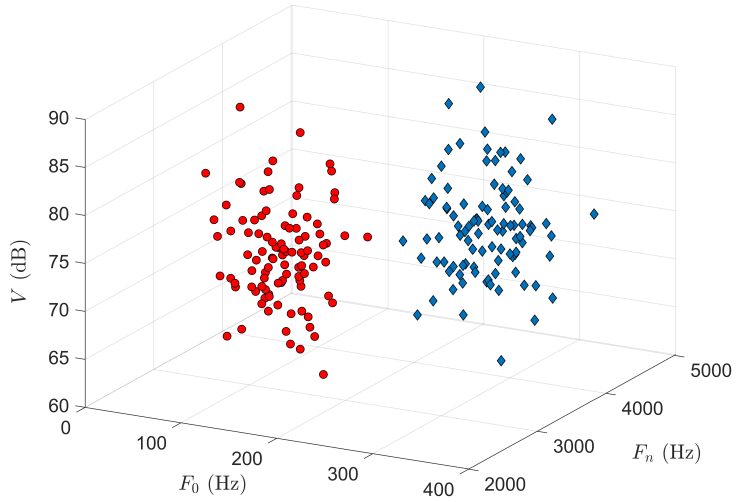


Figure 3.5: Artificial dataset in a 3-dimensional space including relevant and irrelevant features.

contribute to the separation of the classes.

At this point, our problem consists of only 3 features, 2 relevant and 1 irrelevant. Therefore, identifying the relevant ones can be done visually. However, in real-life problems, and in fact for the problems you need to solve here, you will have multiple features available and more than 2 classes. As a consequence, the visual inspection becomes unfeasible.

3.2.3 Feature selection

The methodologies for feature selection can be grouped into two main categories: filters and wrappers. In this module, you will experiment with a filter method called Minimum Redundancy Maximum Relevance (mRMR). As you can already imagine, this methodology finds the most relevant features, while minimizing their redundancy. This methodology is based on formulations of mutual information between the features and the targets y_i. For now, we will not dig into the theory behind this approach but if you are interested, you can read the paper by C. Ding and H. Peng, entitled "Minimum Redundancy Feature Selection from Microarray Gene Expression Data" that you can find on Brightspace.

In this module, you can use the Matlab function `fscmrmr`, as follows:

```
[idx,scores] = fscmrmr(X,Y);

bar(idx,scores(idx)),ylim([0 1])
xlabel('Feature')
xticklabels({'$F_0$','$F_n$','$V$'})
set(gca,'TickLabelInterpreter','latex');
```

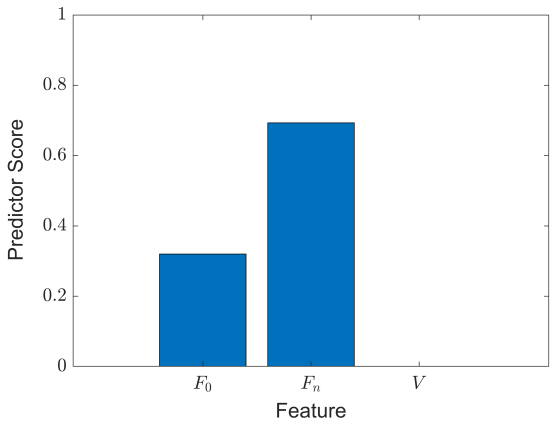


Figure 3.6: Importance of the features calculated using mRMR

```
ylabel('Predictor Score')
```

This will give you the results depicted in Figure 3.6, where it is confirmed that V is not useful for this problem, while F_n appears to be the most informative feature.

The predictor score is related to the importance of each feature for the problem you are trying to solve. The next step is to decide how many features you want to include in the next steps of the algorithm. At this point, you either define a number m of features that you want to keep, hence, you keep only the m features with the highest predictor scores. Another possibility is to keep the features with scores larger than a certain threshold. Keep this two possibilities in mind for the assignments. For this example, we will keep all features with a score larger than 0.2. The selection of this threshold is arbitrary and it is often based on the final performance of your algorithm. We will come back to this in Section 3.2.7.

3.2.4 Training a model

After selecting the most relevant features, the next step is to find an optimal function that can solve the classification problem. There are many algorithms that can be used for this, for instance:

- Linear discriminant analysis (LDA)
- Decision trees
- k-nearest neighbors
- Perceptron
- Logistic regression
- Naive Bayes classifier
- Support vector machines
- Neural networks

Here, we will focus on LDA and decision trees, although you are free to experiment with all the other algorithms (at the end of this module you will know how). The reason to focus on these two types of classifiers is that they are among the simplest approaches that could be used to solve the problems in this EPO4.

Many of the machine learning classifiers are based on distance metrics, which are sensitive to the number of dimensions and the range of values per dimension. For instance, for our example, we have 2 relevant features, one with values from 0 to 400 Hz, and another one with values from 2500 to 5000 Hz. If we use the Euclidiean distance to measure how close the different points are to each other, the second dimension will influence the distance measure a little bit more than the first feature. This becomes a serious problem when we deal with features of different nature. For example, when we use binary features together with features that get values in the range of thousands, the distance metric will be biased towards the second type of features. To overcome this limitation, it is adviced to normalize the features. An often used approach is to first subtract the mean of each feature and divide by its standard deviation. This can be achieved using the function `zscore` in Matlab. For our running example, you can simply use

```
normX = zscore(X);
```

$\rightarrow$ Plot the resultant input space. Do you notice that the feature values are now comparable?

Linear discriminant analysis (LDA)
The main goal of LDA is to find a vector $\mathbf{w}$, so that after projecting the data onto that vector, the following conditions are satisfied:

- largest separation between the projected class means
- minimum overlap between the classes

LDA assumes that each class is normally distributed, with mean $\mathbf{m}_k$ and covariance Σ_k. Graphically, you want to find the direction of $\mathbf{w}$, so that when you project the input data onto it, the means of the classes is maximized, while minimizing the within-class variance. Figure 3.7 illustrates this projection.

Given the training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, with $\mathbf{x}_i \in \mathbb{R}^l$, l the number of features, $y_i \in \{1, \ldots, K\}$, and K the number of classes, the means of the classes are defined as

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{x}_n \qquad \text{and} \qquad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{x}_n, \tag{3.2}$$

for $K = 2$, and N_1 and N_2, the number of samples in class 1 (C_1) and 2 (C_2), respectively. The projected means become

$$\mu_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{w}^T \mathbf{x}_n = \mathbf{w}^T \mathbf{m}_k, \tag{3.3}$$

and the projected within-class variances

$$\sigma_k^2 = \frac{1}{N_k} \sum_{n \in C_k} (\mathbf{w}^T \mathbf{x}_n - \mu_k)^2. \tag{3.4}$$

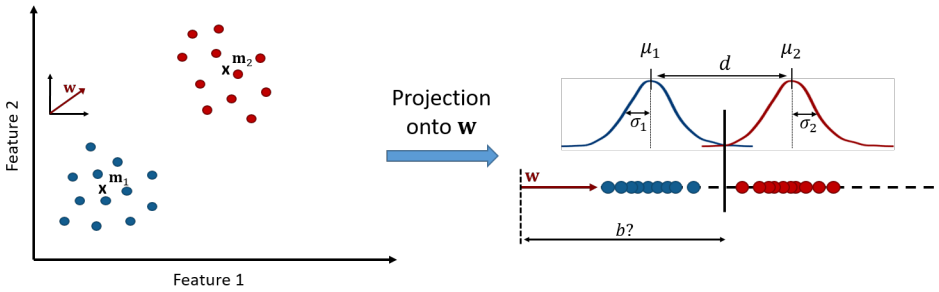


Figure 3.7: Data projection in LDA. The decision boundary is defined in the projected data with a bias value of b from the origin.

The goal is to find the vector $\mathbf{w}$, so that $d = (\mu_1 - \mu_2)^2$ is minimized, and $(\sigma_1^2 + \sigma_2^2)$ is maximized. To this end, we can define the following *objective function*

$$\mathcal{J}(\mathbf{w}) = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{3.5}$$

We can rewrite this as

$$\mathcal{J}(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}, \tag{3.6}$$

where $\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$ and we call the between class scatter matrix, and

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T, \tag{3.7}$$

the within class scatter matrix.

To maximize $\mathcal{J}(\mathbf{w})$, we take

$$\frac{d}{d\mathbf{w}} \mathcal{J}(\mathbf{w}) = 0, \tag{3.8}$$

then $(\mathbf{w}^T \mathbf{S}_B \mathbf{w}) \mathbf{S}_W \mathbf{w} = (\mathbf{w}^T \mathbf{S}_W \mathbf{w}) \mathbf{S}_B \mathbf{w}$, and after ignoring all the scalar values, we find that $\mathbf{S}_W \mathbf{w} \propto \mathbf{S}_B \mathbf{w}$ ($\propto$ indicates proportional to). Again, remember that we are only interested in finding the direction of $\mathbf{w}$, so we can ignore all the scalar values. The solution of LDA is then

$$\mathbf{w} = \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1). \tag{3.9}$$

$\rightarrow$ It is a good exercise to check your linear algebra knowledge, to derive the solution in equation (3.9) from equation (3.5).

After finding the vector $\mathbf{w}$, the only thing that is left to find is the bias b (see Figure 3.7). If the class covariances are the same ($\Sigma_1 = \Sigma_2$), then

$$b = \frac{(\mu_1 + \mu_2)}{2}, \tag{3.10}$$

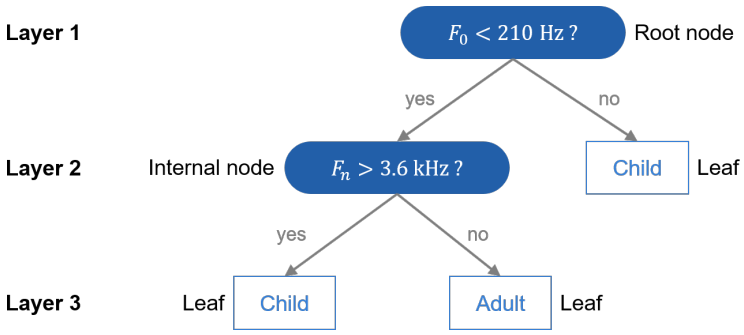


Figure 3.8: Decision tree to solve the simulated example.

and the classification, or the prediction of the class label, becomes

$$\hat{y}_i = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x}_i > b \\ -1 & \text{otherwise.} \end{cases} \tag{3.11}$$

In order to classify the artificial dataset we created before, you need the function `lda`, included on Brightspace. Keep in mind that this algorithm can solve problems that are linearly separable. When the separation between the classes is not evident, or when the decision boundary is not linear, other algorithms like decision trees can be used. Furthermore, this specific function was designed for binary classification problems. For multi-class problems, refer to the Matlab implementation available in the classification learner, explained later in this module.

For a more detailed explanation of LDA, watch the online material offered by StatQuest, available here.

Decision trees

A decision tree is a structure consisting of a root node, internal nodes, and leaves. Figure 3.8 illustrates the decision tree that can be designed to distinguish children voices from adult voices, using the features that were selected as the most significant ones.

$\rightarrow$ For this problem, only three layers are needed to solve the problem, but think of a simpler tree that could also solve this problem. Write down the model for $\hat{y}_i$, based on the tree in Figure 3.8 and on your alternative solution.

Often, deeper trees (i.e. with more layers) are used for non-separable problems, when the problem can only be solved using a nonlinear decision boundary, or with multiple features.

One of the main advantages of decision trees is their high interpretability and efficient learning. These advantages make them one of the most popular methodologies in machine learning.

A very powerful machine learning algorithm, based on decision trees, is called *Random Forests*. It has outperformed classical techniques in solving multiple classification problems and you can use them here to solve the assignments of this module. To learn about random forests, you can watch a 10-minute

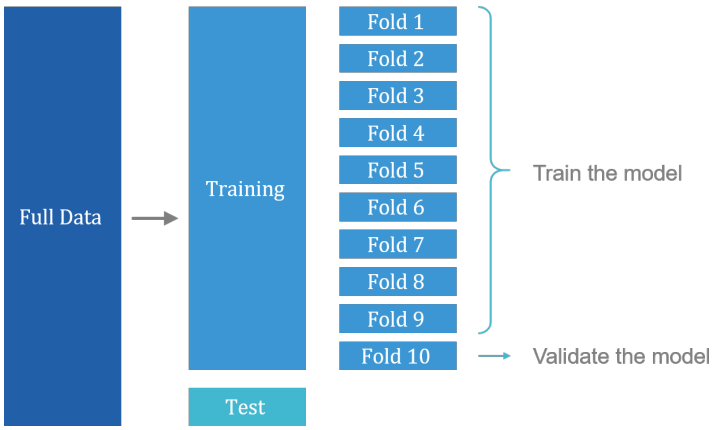


Figure 3.9: Split of the data into training, validation, and test sets. The validation scheme is called 10-fold cross-validation.

video, offered by StatQuest, available here. In this video, the concepts of bootstrapping and bagging are introduced. Please watch this video since it is important to design the random forest in the assignments.

In Matlab, a random forest is referred to as "Bagged trees".

3.2.5 Validation

An important phase when designing a machine learning algorithm is the validation phase. This phase is crucial to find a "good" model that can achieve a good performance on unseen data (i.e., test set). This is called a good generalization. In LDA, the cross-validation approach could be used to find the optimal model: $\mathbf{w}$ and b parameters. Each time, a model is trained on 9 folds and the remaining fold is used to check the performance of the model. You can then select the best model obtained during this cross-validation, as the final model, and perform the ultimate test on the "hidden" test set (See Figure 3.9). For decision trees, on the other hand, cross-validation can be used to adapt the depth of the trees.

There exists different types of validation techniques. On the one hand, it is possible to use a completely different dataset for this step. This requires large amounts of data, since you will need independent sets for training, validation, and test. The most common scheme is called *cross-validation*, where the training set is split into subsets, or so-called folds, as indicated in Figure 3.9. Typically, a 10-fold cross-validation is used to train the models. This means that the training set is split into 10 different folds, and each time, the model is trained on 9 folds and validated on the remaining one. This is done so that the selected model can achieve a comparable performance for all sets. As you can imagine, this is an iterative process and it is often the bottleneck in machine learning. Reasons for this include the fact that multiple models must be evaluated, and that often the "optimal" solution is not unique. You might feel tempted to select the number of folds as high as possible. This approach is called *leave-one-out cross-validation*, but it requires a huge amount of computation time when your dataset is very big. Therefore, the selection of the number of folds is crucial.

3.2.6 Performance evaluation

In classification, the performance evaluation is done based on the number of samples that are correctly or incorrectly classified. These instances are called:

- **True positives (TP):** Samples that were correctly classified in the positive class
- **True negatives (TN):** Samples that were correctly classified in the negative class
- **False positives (FP):** Samples that belong to the negative class but were wrongly classified in the positive class
- **False negatives (FN):** Samples that belong to the positive class but were wrongly classified in the negative class

These values are used to generate what is called the *confusion matrix*, from which multiple performance metrics can be calculated. The most common ones are:

- **Sensitivity:** $Sen = \frac{TP}{TP+FN}$
- **Specificity:** $Spec = \frac{TN}{TN+FP}$
- **Accuracy:** $Acc = \frac{TP+TN}{TP+TN+FP+FN}$

As you can imagine, the best performing classifier is the one with the highest Acc, Sen, and $Spec$.

There are multiple functions that you can use to evaluate the performance of your classifier. For instance, `confusionmat`, `plotconfusion` and `classperf`. Be aware that the input to those functions must be in a certain format. For example, some functions do not allow numerical values, therefore you will need to replace the current numerical labels by categorical variables. You can do this as follows

```
GroundTruth(Y==1) = {'Children'};
GroundTruth(Y==-1) = {'Adult'};
```

Other functions do not accept negative values in the labels.

To use the function `plotconfusion`, you need to transform the vector of labels and the output of your classifier into arrays of size $K \times L$, with K the number of classes and L the number of samples. You can use the following code, for 2 classes:

```
% Plot the confusion matrix
GT = zeros(2,length(Y));
GT(1,Y==-1) = 1; GT(2,Y==1) = 1;

Pred = zeros(2,length(Y_predicted));
Pred(1,Y_predicted==-1) = 1; Pred(2,Y_predicted==1) = 1;
plotconfusion(GT,Pred)
```

Keep this in mind when evaluating your classifiers, and always consult the help of those functions.

3.2.7 Testing a model

Once the best algorithm is selected, the final, and most important phase is performed, namely, the testing of the model on unseen data. At this point, all the performance metrics need to be calculated, and these determine how good your model **generalizes** what it learned from the training set. **You always need to report the performance on an independent test set.** Keep this in mind for the assignments and whenever you use machine learning in the future. By testing your model on unseen data, you really verify that the algorithm is able to deal with new data.

You can use here the function `lda`. This function assumes that both classes were sampled from normal distributions with the same covariance. Therefore, the prediction of the label is performed using Eq. (3.10). For this function you will need a training set and an independent test set.

```
Xtrain = normX([1:N_c/2 N_c+1:N_c+N_a/2],1:2);
Ytrain = Y([1:N_c/2 N_c+1:N_c+N_a/2]);
Xtest = normX([N_c/2+1:N_c N_c+N_a/2+1:N_c+N_a],1:2);
Ytest = Y([N_c/2+1:N_c N_c+N_a/2+1:N_c+N_a]);

[class,ProjectedData]=lda(Xtest,Xtrain,Ytrain);
```

→ Evaluate this algorithm using the functions mentioned in the previous section.

For this example, the classification accuracy is typically 100% because the problem is linearly separable. However, in most of the real-life applications, this is not the case and one might need to allow some errors in the results. There are some techniques that can help finding the best model so that these errors are minimized but they are out of the scope of this EPO4.

There are many things that you can adapt when the classification performance is poor on the test set. You can change the threshold of acceptable features in the feature selection step. You can also go back to feature extraction and think of new features that could be representative for the dataset. This will be important and useful in module 2. Other options are change the type of classifier, change the number of folds, etc...

3.3 CLASSIFICATION LEARNER

In Matlab, there is a tool, called Classification Learner, which allows you to train multiple classifiers. A video tutorial can be found on Brightspace. Feel free to use this app for the assignments, but always justify the selection of parameters, features, folds, classifiers, etc.

3.4 ASSIGNMENTS

There are many data repositories online that are typically used to benchmark machine learning algorithms. One of them is the UC Irvine Machine Learning Repository. It maintains 497 datasets that can

be used for different learning tasks like regression and classification. In this module, you will use the Parkinson Speech Dataset with multiple types of sound recordings. This dataset consists of two subsets of data. Here, you will only use the set called "Train Data". We already prepared it for you and you can download it from Brightspace. The file is called `Data_UCI_EPO4.mat`

3.4.1 Lab tasks

This dataset consists of 40 subjects, 20 patients suffering from Parkinson's disease (Patients in the dataset), and 20 healthy volunteers (Controls in the dataset). For these assignments, it is not relevant to consider the patients separately from the controls. This information can be ignored, but it is mentioned here for completeness.

Each subject was asked to produce the following sounds:

- Sustained vowel **a**
- Sustained vowel **o**
- Sustained vowel **u**
- Numbers from 1 to 10 (N1-N10 in the dataset)
- 10 different words (Word1 - Word10 in the dataset)

Each sound is characterized by 25 different features that are included in the dataset as a matrix of 40×25. To find out how these features were extracted, check the following paper, available on Brightspace:

Erdogdu Sakar B., Isenkul M., Sakar C.O., Sertbas A., Gurgen F., Delil S., Apaydin H., Kursun O., "Collection and Analysis of a Parkinson Speech Dataset with Multiple Types of Sound Recordings", IEEE Journal of Biomedical and Health Informatics, vol. 17(4), pp. 828-834, 2013.

The dataset contains the `Subject_Label`, which indicates whether the subject is a patient or a control. For the assignments, this label information should not be relevant for the problem you need to solve. In fact, the classification should work for both control and patient data.

(Optional) It might be that the performance is lower for patient data. Keep this in mind when evaluating the performance of your algorithm. For instance, you could split both groups and evaluate the performance separately.

For your midterm report, you need to justify every step in the design, for both assignments separately. Start from the separation of training and test set, your validation scheme, the classifier that you select, and the performance evaluation. **Remember to always report the performance on an independent test set**.

Task 1: Words vs. vowels classification The first task is to design a binary classifier to discriminate between one word and one vowel of your choice. Here you can use as a word any number if you find that it can be better distinghuish from a vowel. Ideally, you should be able to separate any vowel from any word, but if you find that one particular word or number leads to the best performance, go ahead and

use it but always justify your selection. This will help you in module 2, when selecting your dictionary of commands.

You can do this by selecting pairs at the time. For example, N1 and the vowel **a** first, then other pair and so on. You could, for example, classify between **a** and all the words, one word at the time, and select which word leads to the best performance. Remember, for this assignment you do not need to select all the words at once, since here you are dealing with a binary problem.

To arrange the data matrix and the labels, check the Hint at the end of this module.

Task 2: Word discrimination Here, you will need to discriminate between the words. In the previous task, you need to solve a binary classification problem, while here you will be dealing with a multi-class problem. Can you find out how to interpret the confusion matrix in this case?

For this assignment, you first need to design a classifier for all words. From the confusion matrix you need to select two words for which the performance was the best. Train the new model with only those two words and test it on an independent test set. It is important to look at the features that can be used to separate between words. This information will be useful in the module of voice processing.

Hint To arrange your data matrix **X** and your vector of labels for a specific task, you can do the following. Here we simply select 2 random features and we assume the problem is to differentiate between the number 1 and word 1.

```
Sel = [1,13]; % Two random features
X = [N1(:,Sel);Word1(:,Sel)]; % Data matrix for the selected features
Y = [zeros(40,1);ones(40,1)]; % Vector of labels, where 0 is the class N1 and 1
    the class Word 1

scatter(X(Y==0,1),X(Y==0,2))
hold on
scatter(X(Y==1,1),X(Y==1,2))
legend({'Number 1','Word 1'})
```

This will generate a dataset like the one shown in Figure 3.10.

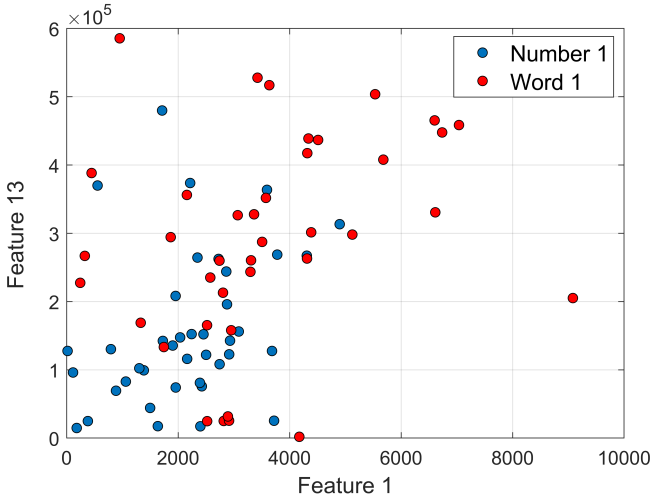
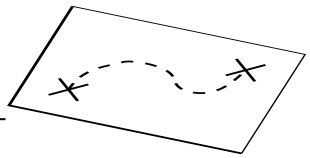


Figure 3.10: Example of 2 random features for the differentiation between the Number 1 and Word 1.

Chapter 4

MODULE 3: VOICE PROCESSING



Contents

In this module, you will learn to process your own speech signals.

First, you will need to record your own audio signals and extract some basic speech features. You will then need to use a pre-recorded dataset available on Brightspace to design a machine learning algorithm to differentiate between commands. The same machine learning pipeline covered in Module 2 will be used here.

You will also need to create your own dictionary of commands. At this point, you will need to consider multi-subject scenarios, words vs. vowels vs. numbers, and different machine learning techniques.

For the mid-term challenge, you will need to create a demo, where you speak to your computer and you use your previously developed model to identify the spoken command. The output of your model can then be used to control the car in the next modules. You can implement your model directly to your car model, or you can use the snake game in the optional assignment.

Learning objectives The following is learned and practiced in this assignment:

1. Extract characteristic features from speech signals
2. Develop a classifier to identify digits from the features extracted using speech processing techniques
3. Create your own dictionary of commands
4. Identify, in real-time, the spoken commands using machine learning

Deliverables As part of the mid-term report:

1. A section describing your experiments (solutions of the assignments), in which you explain your feature extraction, selection, design of the classifier, the performance, and conclusions based on this.

2. Matlab functions:
   - `RemoveSilence.m`
   - `FindPitch.m`
   - `VoiceProc.m`
   - `TestSpeechModel.m`
3. A dictionary of your own commands, used to train your classifier.
4. Trained classifier (`*.mat` file)
5. Live demo

Preparation Make sure you have read this module before the start of the scheduled lab day.

Time duration Two lab sessions; two homework session for reading and to prepare the report.

What is needed

– Brightspace:
  - Words dataset with speech signals: `datarec_directions.mat`
  - Matlab function `FormantsEpo4.m`
  - (optional) Matlab function `SnakeGame.m`

4.1 BASIC SPEECH CHARACTERISTICS

In this section, you will extract two basic features of a speech segment. First, you need to record yourself pronouncing a sustained /a/. You can do this, using the following code:

```
Fs = 8000; % sample rate of the microphone
recObj = audiorecorder(Fs,16,1); % create audio object, 16 bits resolution

disp('Start speaking...')
recordblocking(recObj, 2); % do a 2 second recording (blocking)
disp('End of Recording.');
y = getaudiodata(recObj);
```

To listen to your recording, you can either use `play(recObj)` or `sound(y,Fs)`. If you plot your signal in time, you should get something like the signal shown in Figure 4.1. Whenever you plot time series, always convert your x-axis to time. For the report, **do not plot in samples**.

In order to have the signals that you record at a comparable amplitude, you can normalize them by first subtracting the mean and dividing by its largest value. Remember that there are other forms of normalization, as the one you used in Module 2 for feature normalization.

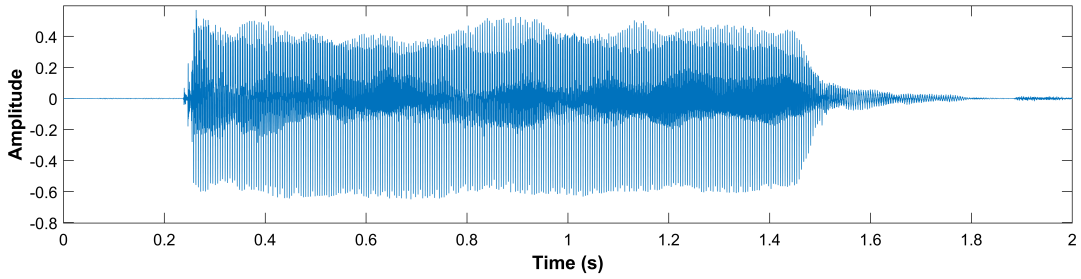


Figure 4.1: Example of an audio recording of a sustained /a/.

As you can see from Figure 4.1, there are some periods of silence at the beginning and end, which should be removed in order to better capture the sound. For the example in the figure, you need to keep the signal from approximately 0.25 s to 1.5 s.

→ Create a function `RemoveSilence.m` to remove those silences automatically. You need to use it for the rest of the module.

Hint You can assume that the signals of interest have an amplitude larger than a certain threshold, so everything below that level could be considered as silence. Keep in mind that this only needs to be done at the beginning and end of the word/sound, so be careful not to remove samples within the word/sound.

Once you have your signal of interest, you can extract the basic acoustic features. In speech processing, the signals are typically analyzed in segments of 20 ms. The reason for this is that we need to assume stationarity, which, as you can imagine, is not satisfied during speech.

There are two main features that you will extract from the speech signals, and they correspond to the *pitch* and the *formants*. These features can be clearly observed from the spectrogram of the signal. In order to calculate the spectrogram, you can use the function `spectrogram` in Matlab, which is based on the short-time Fourier transform. You should be already familiar with time-frequency representations since you used them in the Telecom lab, section 3.4.

To calculate the spectrogram, you need to adapt the input variables of the function `spectrogram`, according to the length of the window you want to analyze. For this module, you can use a Hann window of 20 ms, with a 50% overlap.

Hint The function `spectrogram` uses, by default, a Hamming window but you can change this setting. In addition, use the function in such a way that you do not only get the plot but also the time and frequency vectors (`[S, F, T] = spectrogram(...);`). You will need those to plot the formants and pitch using the same temporal scale. Also, plot the frequency on the y-axis, in Hz, not in normalized frequencies.

Figure 4.2 shows the spectrogram for the signal of interest. It can be seen that the frequency content is somewhat constant throughout time, since the signal corresponds to a sustained vowel.

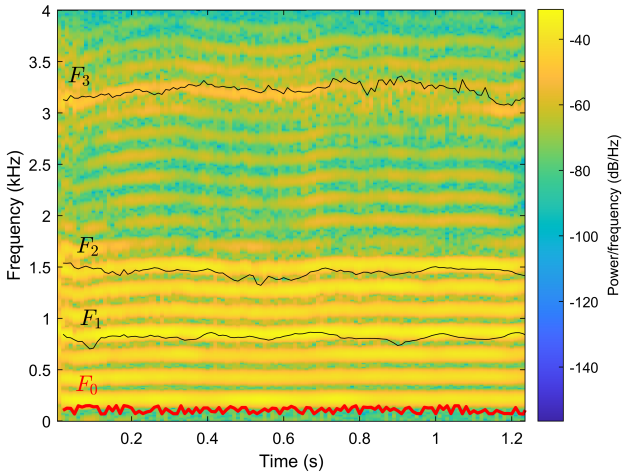


Figure 4.2: Spectrogram of a sustained /a/ with the pitch indicated as F_0 and 3 formants, F_1, F_2, and F_3.

4.1.1 Pitch detection

Here you will need the `Audio Toolbox` of Matlab. To install it, you need to go to `home->add-ons->get add-ons`, and search for *Audio* `->Install`.

Pitch is a subjective feature of a sound that allows to label it as a "lower" or "higher". In fact, it can be described as the fundamental frequency (F_0) of the sound. In order to estimate the pitch, you can use the function `pitch`. Note that this function also requires similar input parameters as the spectrogram, namely, the window length and overlap. Figure 4.2 indicates the variation of the pitch in time. In this module, you do not need to go into the theory behind the different algorithms for pitch determination. You can use here the *Cepstrum Pitch Determination* (CEP) technique.

$\rightarrow$ Create a function called `FindPitch.m`, where you find the pitch for each window of 20 ms, with an overlap defined by `ov` (take 50%). You will also need to plot it on top of the spectrum as indicated in Figure 4.2. Remember to use the same temporal scale that you got from the spectrogram function.

4.1.2 Formant estimation

In Module 2, we briefly mentioned some typical formant values. Here, you will estimate these frequencies using linear predictive analysis, or so-called linear predictive coding (LPC).

First, let's refresh the concept of formant. Formants correspond to the resonances of the different cavities of the vocal tract. When we produce vowel sounds, we adapt the vocal tract (e.g., changing the position of the tongue) in order to change its shape so that multiple reflections of sound are produced. These multiple reflections generate what is called the vowel formants, F_1, F_2, and F_3. Keep in mind that formants are not only present during vowel production, in fact, any sound that we produced is accompanied by a certain collection of formants. However, the formant vowels, and their relationships have been clearly defined

and widely studied in literature.

A demonstration of formants can be found here. As you can see in the video, formants can be derived from the spectral envelope of the audio signals. This can be done using LPC, which uses concepts like autocorrelation and model estimation. You should be already familiar with the concept of autocorrelation but you will know more in the course EE2S31. For more information we refer to Section 4 in Signal Processing Supplement, Roy D. Yates& David J. Goodman (2014).

To estimate the formants, you can follow the procedure describe here. This procedure is included in the function `FormantsEpo4.m`. The function contains the different steps to calculate the formants in windows of length `L`, with an overlap defined by `ov`, see `for loop` of line 27. You can use a similar loop to calculate the pitch in a moving window approach. For instance:

```
for iter = 1:ov:(N - L)
    count = count+1;
    tmp = y(iter:iter+L-1,:);

    % Find Pitch
    % ptch(count) =
end
```

The estimated formants of the sustained /a/ are indicated in Figure 4.2.

→ Create a function that estimates the pitch and formants of the signal of interest (i.e., signal without silences). You can use the following code as a starting point:

```
function [t_vec,f0,frmtns] = VoiceProc(y,Fs,ShowPlot)

%% Remove the silences before and after the signal of interest
x = RemoveSilence(y);

%% Pitch estimation
f0 =

%% Formant estimation on the shortened signal
[t_vec,frmtns] = FormantsEpo4(x,Fs,L,ov);

%% Plot the spectrogram with the features
if ShowPlot == 1
    [S,F,T] = spectrogram(x...);
    hold on
    % Plot the pitch
    % Plot the formants
```

```
end
```

4.2 FEATURE EXTRACTION OF SPEECH SIGNALS

Once you calculate the pitch and formants of the sustained /a/, you can process the dataset available on Brightspace, `datarec_directions.mat`. This dataset contains audio signals produced by two speakers, one woman and one man. Four words were pronounced 10 times, up, down, left, and right. The spectrograms of one example per word are shown in Figure 4.3, with their corresponding pitch and formants. You can already observe the complexity of the different sounds in terms of the organization of the formants. For instance, you can identify the segments where vowels are pronounced. Keep this in mind when generating your own dictionary.

In Module 2, we mentioned that the pitch and formants are different for women and men. You can prove this using the codes you developed in the previous sections, and applying them to this dataset.

Now, you need to extract features per word, so that you can use them to discriminate between them. Below you can find some features that we suggest but feel free to create your own features. As long as it works to separate between the words, you can use them, but always justify your selections and findings.

In Figure 4.3, it is clear that the variance of the formants in the word **Up** is much lower than the variance in the other words. A different feature that could be used is the median of all formants. Figure 4.4, shows the median and variance of F_1 for all words produced by one speaker in the dataset. You can see that there is already some separation between the words (i.e., classes). Remember that this was computed using only the signals of interest. If you include silences in this calculation, your features might look very different.

$\rightarrow$ Generate the feature plot for both speakers and see if there is an overlap between the words for either of them.

From the results that you obtain on this dataset, you can define the following:

- Which sounds to include in your dictionary?
- Which team members should be giving orders to the car?
- Which features can be relevant for the problem?

You can extract these and other features that you can design from the dataset. After that, you can use mRMR, as studied in Module 2, to select which features are the most important and relevant to solve your classification problem.

Hint You can try to find the vowels within the words, by searching for a region where the formants are "stable".

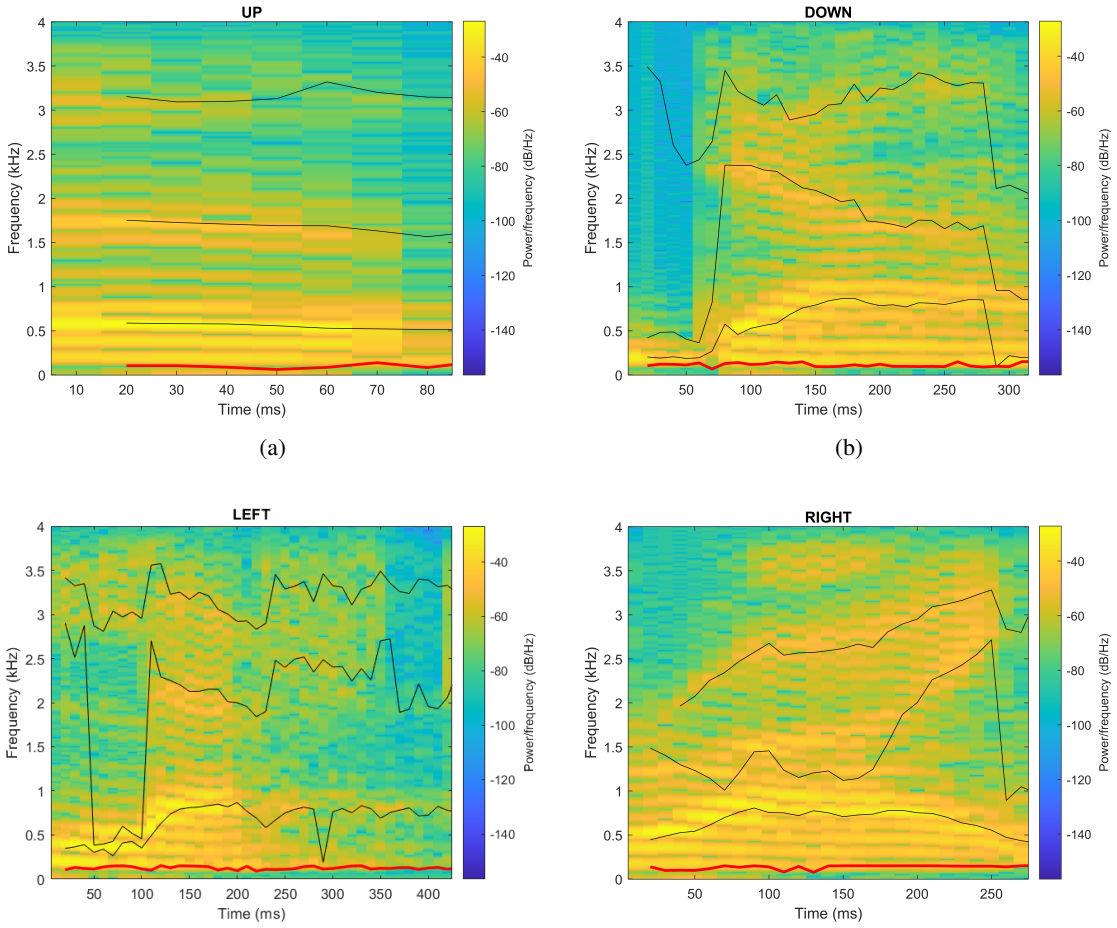


Figure 4.3: Spectrograms of 4 different words of the `datarec_directions.mat` dataset.

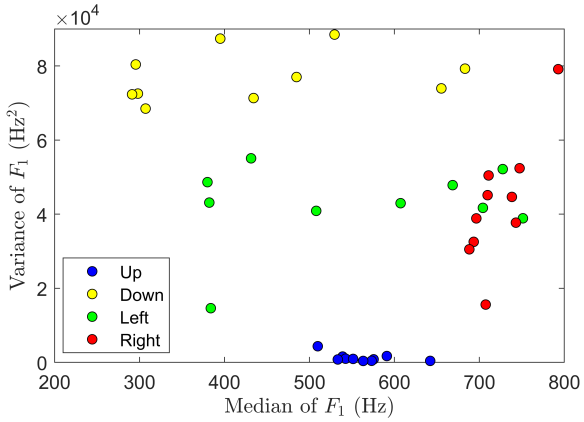


Figure 4.4: Median and Variance of F_1 for one speaker.

4.3 ASSIGNMENTS

For these assignments, you will need to use the dataset `datarec_directions.mat`, and you will need to create your own dictionary of commands, which you will use to train your classifier.

Some general things to keep in mind when processing the audio signals are:

- Only remove the silences at the beginning and at the end of the word
- Adapt and create all the Matlab functions so that they work for any audio signal, since you will need to use them for the final challenge
- Remember that the methods covered in this module to extract the speech properties, assume stationarity. Therefore, it is important to use the moving window approach.
- You always need to create a training set and keep aside an independent test set. The latter, and only the latter, can be used to report performance.
- Since you need to normalize the features only using the information of the training set, you need to compute the mean and standard deviation (std) of the features in the training set. Save those values and use them to normalize the features in the test set and during the demo. Remember that the `zscore` function subtracts the mean from the features and divides them by the std. Therefore, you will need to do this yourself for the new data points, using the mean and std of the training set.

4.3.1 Lab tasks

Task 1: Feature extraction and classification of the speech dataset
$\rightarrow$ From the pre-recorded dataset, compute the mean, median, and variance of the pitch and each formant for each word separately. Also, compute the ratio F_1/F_2 and its variance. Do this for both speakers.

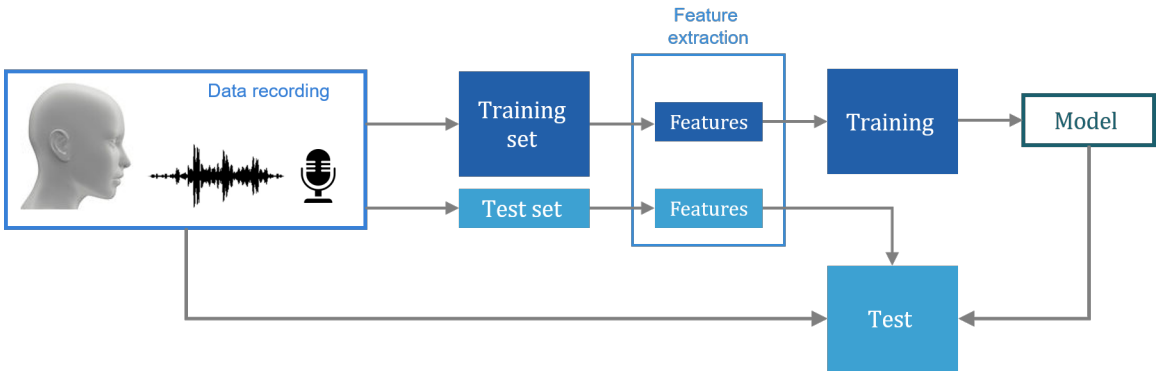


Figure 4.5: Speech processing pipeline for task 2

→ Be creative and generate your own set of features to characterize the words.

→ From the full set of features, use mRMR to find the most discriminative features and generate a 2D plot with the most relevant features for both speakers combined

→ Perform classification. You can use the `Classification Learner app` or any other function you want. Always justify your selection. Select your final model and report its performance on a test set, that you selected.

→ Answer the following questions:

- Can you have a comparable performance for the man and the woman in the data set? Note that you do not need to train one model for the man and a different one for the woman. In principle, one model should be general enough but if you test it on only samples produced by the man, the performance might be different. Is this the case?
- What kind of sounds are easier to distinguish?
- Will the performance improve if you only select two words? If so, repeat the procedure with only those 2 words and report the performance.

Task 2: Create your own speech processing block

For this task, you need to implement the pipeline indicated in Figure 4.5. Based on the performance in the previous task, you can identify which words or sounds could be used to give commands to the car. Here, you need to create your own dictionary consisting of at least 3 commands. Those commands can be any sound or word, but you must include at least one word (any word). Here you need to include as many repetitions as possible of the same words/sounds. Remember, the more the better, since your machine learning algorithm performs best when trained on big data sets. If you decide to record 100 repetitions of each command, you can use 70% of the data for training and keep aside the remaining 30% for testing.

Another important thing to consider is the microphone and the person giving the commands to the car.

You can select at least two persons of the group, or you can design an algorithm that is subject independent. At the end, the instructors will test your algorithm and the model should be as general as possible.

Hint It is always nicer to have as many speakers as possible. As a result, you will train a very general algorithm and you will find the sounds and features that make the commands separable.

After you create your own training set, you need proceed with feature extraction. Here you can use the features you found most discriminant in the previous task, or you can perform feature selection again. Report on the variability of your findings, in case you find different features to be more discriminat. After that, train and select your classification model.

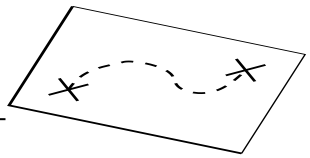
Use your test set and report the performance of your model. In addition, you need to test your model in real-time. To this end, you need to generate a function called `TestSpeechModel.m` consisting of the following parts:

```
%% Create an audio object and record a word/sound

%% Segment the signal of interest

%% Feature extraction

%% Load your model

%% Test the model

%% Display the command
% For instance: disp('You said XXX');
% or disp('The command was XXX');
```

Task 3: Command execution (optional)
In case your car model does not respond to the voice commands, use the function `snake_game.m`, available on Brightspace. Replace the key commands by your voice input and use it to demonstrate the performance of your model.

Chapter **5**

MODULE 4: VELOCITY CONTROL



Contents

In this Module, we give some basic ingredients to model KITT as a dynamical system. This forms the underlying model for the virtual model that emulates the real KITT, and you can use this to predict its behavior.

For the mid-term challenge, we will only drive the car on a straight line, and the objective is to reach a specified position.

For the final challenge, we will be driving on a 2D plane, where estimates of the (x, y) location of the car are obtained from the audio localization system that will be developed in Module 5. The system model of the car is much more involved now, as it also contains the steering angle; in fact it is a non-linear "non-holonomic system" (see Module 6). The augmented model also contains time delays, emulating delays for sending the commands to the car. Your challenge will be to make this work!

Even when driving straight and omitting communication delays, the practical control problem is more complicated than what is studied in the EE2S21 course:

– The control input (car velocity setting) has a limited range (discrete steps, quickly "saturates") and also has a dead-zone.

– The system response is battery-dependent (i.e., slowly time-varying). Luckily we can observe the battery status and could try to compensate for that.

However, for the mid-term, we will omit these effects.

We first develop a simple *theoretical* driver model in Matlab/Simulink, that controls the acceleration and brake. Based on this model, we play with a simple controller where the goal is to let KITT race from an initial position to a given final position in shortest possible time.

Learning objectives You will go over a system modeling exercise and design a bang-bang controller.

Deliverable As part of the mid-term report: A brief discussion on the assignments, and a working controller.

Preparation Make sure you have done the following before the start of the scheduled lab day:

– Reading this Module, in particular Sections 5.1 and 5.2;

– Homework Tasks 1 and 2 in these sections.

Time duration Two lab sessions (including system integration); two homework sessions.

5.1 KITT DYNAMICS MODELING

We start by driving on a line, and consider a simple model of longitudinal KITT dynamics, described by Newton's second law. The car's motion is influenced by the following three forces:

- Accelerating force F_a due to the torque delivered by the engine.
- Braking force F_b due to the brakes. This force clearly opposes the motion and decelerates the car.
- Force F_d due to the air drag and other types of friction. This force also decelerates the car and depends on the velocity v as
$$F_d = bv + cv^2.$$

Assume the following car parameters (not guaranteed to be realistic!):

- Mass of the car: $m = 5.6$ kg
- Viscous friction coefficient: $b = 5$ Nm^{-1} s
- Air drag coefficient: $c = 0.1$ Nm^{-2}s^2
- Maximum acceleration force: $F_{a,\max} = 10$ N
- Maximum braking force: $F_{b,\max} = 14$ N

Homework Task 1

1. Indicate the forces and their directions in Figure 5.1.
2. Write the differential equation describing the longitudinal car dynamics. We will numerically integrate this equation in Simulink and we are interested in the evolution of the car position $x(t)$ and the car velocity $v(t)$ over time.
3. Check if the air drag is relevant for our setting. Assuming it is not, you can easily solve the differential equation in closed form.

Figure 5.1: Indicate the forces and their directions.

4. Construct the block diagram representing the above differential equation. Use the following blocks: integrator, sum, gain and square function.

5. Implement the block diagram in Simulink. In the block parameters dialog enter appropriate symbols for the variables, rather than the specific values defined on the previous page (e.g., m for the car mass, c for the drag coefficient, etc.). Write a script called `KITTParameters` in which these variables are assigned their values. Note: use the Simulink scope parameters dialog to set the number of axes in the scope to 3, so that you can plot the acceleration, velocity and position underneath each other. Save the block diagram (Simulink model) under the name `KITTDynamics`.

6. Set the 'Max step size' parameter of the solver to 0.01 second. Set the initial velocity $v(0)$ to 1 meters per second. Simulate the model for 8 seconds using the maximum acceleration force $F_{a,\max}$ and check the acceleration, velocity and position signals in the scope. Interpret the result. Does it correspond to what you would expect from an accelerating car?

7. Add to the Simulink model the block 'To Workspace' in order to export the time vector and the car velocity signal to Matlab (set the 'Save format' parameter in the dialog to 'Structure with time'). Simulate the model again for 8 seconds using the maximum acceleration force: $F_{a,\max}$ and write a Matlab script `KITTVelocity` that will a) plot the velocity as a function of time (label the axes properly) and b) calculate in how many seconds the velocity reaches (or just exceeds) a given speed, say 3 m/s.

5.2 BANG-BANG CONTROL

We will use the model of longitudinal KITT dynamics that you have developed in the previous assignment, and augment it with a controller.

Homework Task 2

1. Open the model `KITTDynamics` you developed previously and save it as `KITTDynamicsFinal`.

2. Simulate the KITT dynamics for braking from the initial velocity of 3 m/s. Set the acceleration force to $F_a = 0$ N, the braking force to $F_b = F_{b,\max}$ and the simulation time to 5 seconds. Does the result correspond to what you expect from this simulation? More specifically, does KITT stop after several seconds of braking? If yes, you can continue to 4. If not, you will need to correct the model. Spend first a few moments thinking about what is wrong in the model and how it can be fixed taking the physics of braking into account. If you cannot figure it out yourself, go to 3.

3. A typical car brake generates the braking force by pressing the brake pads against the brake disk which is mounted to the wheel. The resulting friction force decelerates the car. As the friction is mainly dry (Coulomb) friction, the absolute magnitude of the friction force will be independent of the car velocity. However, the sign of the friction force depends on the sign of the velocity, as shown in Figure 5.2.

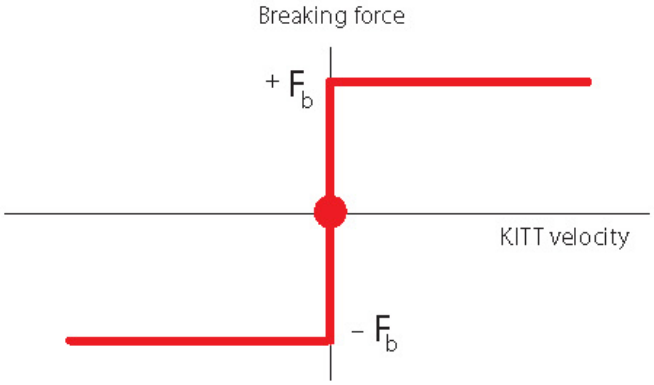


Figure 5.2: Sign of friction force

To implement this model, we can use the block 'Sign' from the 'Math Operations' library. However, because of the finite numerical precision, the simulation with the sign function alone will not work well. It will result in frequent switching between F_b and $-F_b$ once the velocity approaches zero. This can be prevented by using the 'Dead Zone' block from the 'Discontinuities' library. In this block you can set a small range of velocities that will give a zero output (say a range -0.001 m/s to $+0.001$ m/s). Finally you will need the 'Product' block from the 'Math Operations' library to implement the product of the sign with the braking force.

4. To keep the system implementation neat, let us first encapsulate the car dynamics model into a subsystem. Make sure the acceleration and braking force input blocks (constants) are on the left and the scope on the right of your block diagram. Using the mouse, draw a selection rubberbox around all the blocks between the force input blocks and the scope. Right-click on of the selected blocks and choose 'Create Subsystem from Selection' in the pop-up. After labeling the 'Inports' and 'Outports' you should end up with a block diagram like that depicted on Figure 5.3. Save this block diagram under the name `KITTRacing`.

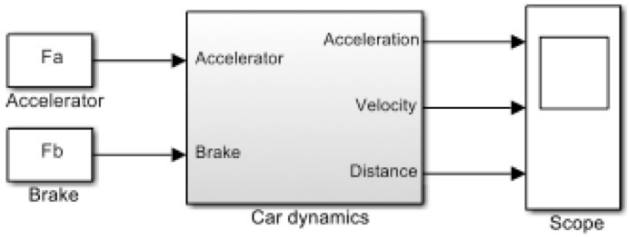


Figure 5.3: Car dynamics Simulink block model

5. Now we are ready to do the main task: implement the racing driver, who will control the accelerator and the brake so that KITT travels a given distance, say 6 m, from standing still to standing still as quickly as possible. More precisely: starting from initial velocity $v(0) = 0$ m/s and initial distance $x(0) = 0$ m, KITT must reach the final distance $x(t_f) = 6$ m at velocity $v(t_f) = 0$ m/s at minimal time t_f. Spend a few moments thinking about how to solve this problem. In any case, you will need to do some calculations in a Matlab script. Call it `KITTDriver`. To run Simulink simulations with `KITTDynamicsFinal` from the script, use the Matlab function `sim`.

Hints

It is easy to see that in order to reach the final position as quickly as possible, the car has to maximally accelerate for as long as possible and then, from a certain moment on, brake as hard as possible. If the point at which the switch from acceleration to braking is correctly calculated, the car will stop exactly at the desired final position. The problem can be reduced to calculating the position (or time instant) at which the driver stops accelerating and starts braking. There are (at least) two different methods to find that point using our Simulink model `KITTDynamicsFinal`.

(i) Create a graph of velocity vs distance, in which you plot acceleration and braking curves. Then find where these curves intersect, as shown in Figure 5.4. The distance coordinate of the switching point is the distance at which the driver must switch from accelerating to braking.

To generate the curves, simulate `KITTDynamicsFinal` twice, once for the acceleration from zero speed and once for braking from a sufficiently high initial speed. To calculate properly the intersection, you need to shift the braking curve so that it ends at the desired final distance. Make sure you run the simulation long enough and from high enough initial speed, so that the curves actually intersect.

To calculate the intersection, create a function `intersectCurves`. You can use Matlab's function `polyxpoly`.

(ii) Alternatively, discretize the distance between the initial and final position with some reasonable resolution (such as 1 meter) and run simulations for the switching points at each of these distances.

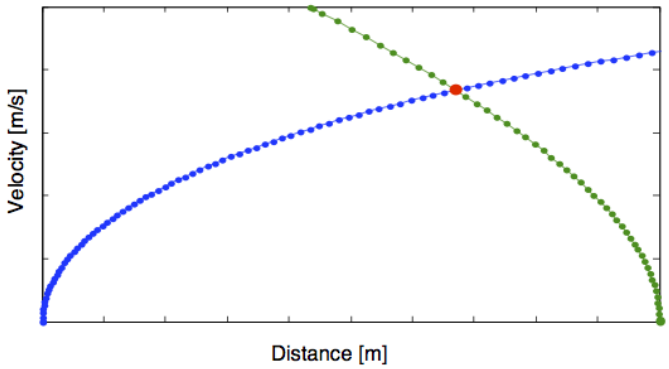


Figure 5.4: Acceleration vs. Braking curves

Select the point for which the car stops at or just past the desired final position. This is a less elegant, brute-force method, but it will probably require fewer program lines than method (i).

(iii) If you are brave, you can also try to calculate this mathematically in closed form.

To implement the driver in Simulink, use the 'Compare To Constant' block from the 'Logic and Bit Operations' library.

5.3 ASSIGNMENT

In the homework tasks you have learned how to control a simple *theoretical model* of KITT using a so-called *bang-bang control* principle, a result derived in optimal control theory. KITT was modeled a point mass, subject to various types of friction, by a time-invariant system. If we want to apply this control principle to the real KITT, we would need an accurate model structure and to estimate the parameters (i.e., *model identification*). However, it is very difficult to obtain an accurate model for the dynamics due to the nonlinear character of the car. Also in practice battery conditions do not remain constant, the car dynamics are then time-varying.

Another complication is that KITT has no brake! But you can apply a negative acceleration force. The difference is that the car will move backwards if you apply it for too long. Otherwise, without force, the car will stop due to friction, but that can take a long time.

When using `EPOCommunications`, the motor settings range from 135 till 165, with 150 being "neutral"; you can assume that 165 corresponds to $F_{\max}$, and 135 to $-F_{\max}$. For bang-bang control, we only need these three settings.

Task 1

To start, we measure acceleration and deceleration curves. We assume a constant (e.g., fully charged) battery voltage.

The control input is the speed setting that we transmit to KITT. The system output is the position and velocity (derivative of position). The observations are the distance of KITT to the wall.

1. Make sure the batteries remain at a constant voltage during this experiment.
2. Start/stop at different speed settings (applied during fixed time intervals) and generate the associated acceleration/deceleration velocity–distance curves. Also record the battery voltage at each experiment and make sure these values are approximately the same for each experiment.
3. Do you reach a constant velocity at some point? After what time?

A complication is that the sensor only measures distance; if you also require a velocity estimate, you have to derive it from the distance observations by taking differences. (This is not as easy as it appears. If you sample slow, to what moment in time does this velocity estimate correspond? And if you sample fast, then the velocity estimate becomes very sensitive to noise.)

Suggestion: If you solved the differential equations by hand (ignoring the drag, which is very small anyway), then you can try to fit this solution to your measurements. Can you find settings for the $F_{\max}$ and friction parameters such that you obtain a good match?

The acquired data will form our "system model".

Task 2

3. Make sure the batteries are charged to the same level as during the model measurements.
4. Calculate for the predefined speed setting and the predefined travel distance the intersection point. What is the corresponding switching time?
5. Accelerate KITT to a predefined desired speed setting and then let it automatically stop at a predefined distance from the starting point.

In the above test, we do not need to make distance observations, it is sufficient to have the model curves and an accurate timer (Matlab). This is an "open loop" solution and not considered adequate in practice.

Task 3

The next assignment is to accelerate KITT to a constant speed and to stop at a predefined distance from the wall. This means you have to incorporate the information from the distance sensors into your control scheme. You will also need the deceleration curve.

For this experiment, make sure that KITT starts accelerating far enough (> 3 m) from the wall to reach a *constant* speed. An estimate of the 'minimal' starting distance from the wall can be computed from the earlier obtained acceleration data.

1. Make sure the batteries are charged to the same level as the model measurements.
2. Accelerate KITT to a constant speed and then let it automatically stop at a predefined distance from the wall.

Task 4

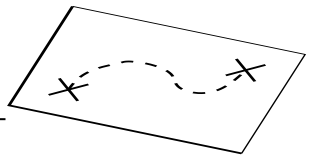
To minimize travel time, we brake by giving a negative speed setting. Obviously that setting should not be maintained very long or else the car will start to drive backwards, which is not allowed. So instead of a single switching time, we need two switching times. Can you devise a suitable controller?

One option is to drive backwards for a predefined amount of time (which depends on the speed of the car).

A practical problem may be that the wheels may start to skid, increasing the nondeterminism of the system.

Chapter 6

MID-TERM CHALLENGE



Contents

Now that you have a working car, can drive it, have voice control, and know how to stop precisely at a specified location, it is time to see how well you did. This is done in this mid-term challenge. It consists of a demonstration and a mid-term report where you document your designs.

Demonstration For the mid-term challenge, show the capabilities of KITT regarding the control system and the anticollision system. Your performance contributes to your final grade.

Report A mid-term report explaining your design with its technical details. This must contain the design choices of your designs so far, the expected capabilities of your system, the problems you faced with linking the parts together. Also answer the specific questions asked in the previous two modules. Suggested length: 15 to at most 20 pages (excluding the Appendix that lists the Matlab code).

The report is prepared as a group. The report is graded and contributes to your final grade.

Preparation Completed Modules 1, 2, 3, 4.

Time duration Part of a lab session for the demonstration, two homework sessions for report writing.

6.1 DEMONSTRATION

Before you can start with the mid-term challenge, it is necessary to have working subsystems as developed in the previous Modules. If your group does not have working subsystems, make sure to contact the practicum coordinators for advice.

Your design must demonstrate a control system that allows KITT to start driving (on a straight line) and stop at a specified distance from the wall.[1] You earn points for speed and for accuracy on the achieved final distance.

[1] Distance is measured relative to the front bumper of the car.

The exact challenge depends on whether you implement voice control or not.

6.1.1 Without voice control

After the GUI is started, the car is not in motion, and it waits for you enter a desired stopping distance and press the *start* button.

You give two demonstrations:

Challenge A Start at about 5–6 meters from the wall (out of range from the distance sensors);

Challenge B Start at about 3 meters from the wall.

In both cases, the exact starting distance to the wall is not known to you. The final distance is specified to you in centimeters and is between 30 and 50 cm.

For each Challenge, you get two opportunities, and the best trial counts.

6.1.2 With voice control

After the GUI is started, the car is not in motion, and it waits for you enter a desired stopping distance. After that, only voice commands are allowed to be given.

You give one demonstration:

Challenge A Start at about 5–6 meters from the wall (out of range from the distance sensors).

The exact starting distance to the wall is not known to you. The final distance is specified to you in centimeters and is between 30 and 50 cm.

Your control system is capable to recognize three separate commands:

– *Ready*: the system initializes and the car responds by an audible signal from the beacon loudspeaker.

– *Go*: the race starts.

– *Stop*: only after this command is recognized, the car is allowed to stop. However, it should determine itself what is the best moment to do so.

The vocabulary of your speech recognition system should consist of at least these 3 separate commands. You are allowed to design your own "language" for these commands.

To facilitate the implementation, you are allowed to press a key on the keyboard so that the system knows to listen to the microphone.

For this Challenge, you get two opportunities, and the best trial counts.

6.1.3 The GUI

The GUI should have an input field to enter the specified stopping distance.

Without voice control, it should have a *start* button. With voice control, it should show the recognized (decoded) commands.

You are not allowed to physically drive backwards (although you can give negative speed commands to implement a hard brake). After you stop, the GUI has to show the position of the car so that the final distance to the wall may be assessed.

The GUI should also show a timer that measures the duration of the race, in the format ss:cc, with ss the seconds and cc the centiseconds.

(Optional) A speedometer or other gimmicky additions that make the GUI more attractive.

6.1.4 Grading

The maximal grade is 100 points. Criteria for judgement are:

– (55 pnt) Ability to reach the goal within 20 cm.

– (max 20 pnt) Accuracy of the final position (in steps of 5 cm). Teams with the most accurate position get bonus points.

– (max 10 pnt) Total time to reach the goal. Teams with the minimum time get bonus points.

– (voice control, 10 pnt) Accuracy of the speech recognition: are the spoken commands decoded correctly?

– (no voice control) The performance on both distances get averaged.

For each Challenge, you receive 2 attempts, the best attempt counts.

Crashing the car against the wall or visibly driving backwards incurs a foul.

6.2 MID-TERM REPORT

The mid-term report explains your designs with its technical details. This must contain the design choices of your designs so far, the expected capabilities of your system, the problems you faced with linking the parts together and a brief summary of the previous modules comprising the relevant conclusions. Suggested length: 15 to at most 20 pages.

Aspects on which the report is judged are:

- *Correctness and completeness of the report:* Theoretical justification and accuracy of the results, addressing all requested tasks. Bonus for taking the analysis/design beyond the strict scope of the

related Module, in particular for introducing innovative solutions, highlighting implications and/or applications to other areas; penalty for unacceptable conceptual mistakes.

- *Quality of the submitted report:* Conformity to the requirements of a scientific report (adequate use of equations, figures, citations, cross-references), readability, layout. Penalty for late submission and for fraud (plagiarism). Remember the format is of that of a technical report, and not that of a homework assignment.

- *Planning* and *teamwork* are also judged. The report should also contain a section that clearly describes these aspects.

The deadline for submission is listed in Brightspace. Submit your report using the corresponding submission folder. Please use a filename that starts with your group number. Indicate the group number and all group members in the comments field.

The report should contain the results of the various assignments, embedded in a natural way, as these will motivate your solution.

The report should be independently readable by someone not familiar with the project, but you can refer where needed, so don't copy large parts of this manual but summarize with your own words and refer to it. The modules suggested questions that you can answer in your report —do this in a natural (self-contained) way. You can also refer to additional literature that you consulted. As mentioned, you should try to be as concise as possible, and judge yourself what is important to be included.

Some other general hints are given next.

How to write and structure your report

Here are some general hints on the structure and contents of your report (mid-term and final report).

- Keep a clear structure and writing style. Make sure you give sufficient factual information (in particular if things don't work).

- *Cover:* Make sure you specify names, study number, group number, date.

- *Introduction:* First determine who is the reader (in this case, the course coordinator, or evaluating committee member, later in life e.g., your boss). The report should be at his level: find an appropriate balance between context details and conciseness. In general, the introduction of a technical report should present the context and describe the design objectives (problems to be solved), at a sufficiently high level. After the problem definition/requirements and an initial analysis that identifies the critical design issues, a typical report would split the problem up into sub-problems (subsystems) which are defined in general terms in the Introduction, and developed individually in the subsequent sections. Sometimes, after the introduction a section is needed that describes the background in more technical or mathematical detail, develops a data model, or defines the measurements and problem statement in more mathematical detail. For the mid-term report, that

structure is perhaps overkill (the subsystems have already been defined for you, and the mathematical models are probably not needed to define the problem sufficiently accurate) so you can be brief.

In general, an introduction may also contain a literature overview, references to similar work, e.g., how similar problems have been solved and what are the limitations of those solutions (thus motivating your present work.) The sections on subsystems may follow the same structure but at a more detailed level.

Place your report in context: "This report describes $\cdots$". You don't need to motivate the project.

- *Sections:* Place each module in a separate section. You don't have to repeat everything from a module, but give sufficient context to make your report independently readable. Embed the assignments in a natural way, and give them intelligible names (not "Task 1").
  - Specifications: What is the objective? What is given already? E.g., schematic, state space equations – you may also define notation here.
  - Analysis: What are the one or two key problems that drive the design? How can these be addressed?
  - Design: What needs to be designed? What is your approach? Make clear what was already given and what is added by you.
  - The resulting design. This could comment on the implementation, refer to the main variables that you use, the setup of a timing loop or interrupt system, etc; generally after reading this, a reader should be able to quickly grasp the Matlab code in the Appendix.
  - Verification: How do you test it is functional? What did you measure/observe? Present your results (e.g., measurement results, a Matlab plot), describe what you see in each plot, and then what you can conclude from this. Don't be naive: things don't work exactly how you design them. Do the debugging systematically. Be sure the plots have correct labels on the axis, and a legend on the line types if you use multiple lines in a single plot. Check the font size; the plot should be readable.
  - If something doesn't work, what is your hypothesis on the problem? How would you verify that hypothesis?
  - It is certainly not appreciated if you only say "It works" or "It didn't work" without documenting this first (i.e., show results/plots and discuss what is seen).
  - It is highly appreciated if you include an analysis that explains how accurate your system could be (or will be), in view of hardware limitations. This analysis is then backed up by the verification stage.
- *System integration:* Apart from the integration aspects, this is a natural place to describe the GUI, and its functionality. Include a screenshot.

  Also in this section, you need a subsection on Verification.

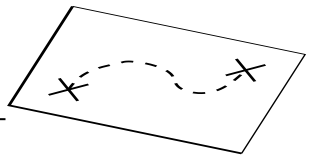
- *Conclusions:* Clearly mention if something doesn't work. This is still the formal part of the report, so keep the language sufficiently formal until this point (no "Tante Bep" stories).
- *Discussion:* Information about the (group) process and any other useful feedback. Your original planning (work division over team members and time) and the actual outcome.
- *Teamwork and Planning:* Describe the way you work together and how you plan and manage the project. See the Kickoff document: how did that work out? Did everyone contribute?

  The material under Discussion and Teamwork can be more informal. In a real (formal) report like a thesis, these sections would be omitted or worked into an Acknowledgement or Postscript.
- *Appendix:* Include Matlab code and/or other details on your design that may be helpful, for example:
  - Properly document and comment input and output parameters.
  - Include author and date information (version, history).
  - Describe data structures and other global variables in detail, i.e., describe them to all levels.
  - Functions should have a header block that explains what the function does and what the input/output parameters are.

  Your appendix has to be sufficiently complete such that the experiments are reproducible by others.

Chapter 7

MODULE 5: LOCALIZATION USING AUDIO COMMUNICATION



Contents

In this Module, the objective is to localize the car using a beacon signal that it transmits. The beacon signal is an audio signal, and its waveform is programmable. The transmitted signal is received by four or more (virtual) microphones, and the received signals are sampled in your PC and made available in Matlab.

In the tutorial in Appendix C, it is shown how if we know the distance of the car (audio beacon) to at least 4 microphones with known (x_i, y_i) coordinates, we can estimate the (x, y) coordinates —and hence the location– of the car. The algorithm finds the location by solving a linear system of equations.

Optionally, we have a chance to use the "Multi-Dimensional Scaling" (MDS) algorithm, also presented in Appendix C. This algorithm can find the locations of the microphones based on estimates of the distances between all pairs of microphones. This is useful if it is unpractical to measure these locations by hand.

Learning objectives Location estimation algorithms: Estimating time differences of arrival (TDOA) using multiple microphones, and then location from the TDOAs. Introduction to the MDS algorithm. This illustrates several signal processing tools, and is a nice application of what you learned in linear algebra (eigenvalue decomposition).

Deliverable In your final report, show Matlab plots and results, and brief discussions on them (Matlab code in an appendix).

Preparation Read the chapter. Also study Appendix B "Programming the audio beacon" and Appendix C "TDOA localization". Optionally, study Appendix C.2 "Tutorial on the MDS algorithm".

Time duration Two lab sessions and 2 homework days.

What is needed

- PC or laptop with 2 microphones, cables and plugs to insert them into the stereo input of the PC.
- Audio beacon (the one built in KITT).
- From Brightspace: Matlab DLL includes for the multichannel ADC: `pwavplaya.mex64`, `pa-wavrecord.m`.
- Ruler or measurement tape (1 to 2 meter).
- A USB memory stick for storing measurement data (in case the PC has no internet connection).

You will work on the desk with 2 microphones, but you will also do some measurements using the experimental set-up with 5 microphones and a PC with an 8-channel sample card.

7.1 OVERVIEW

In this Module four algorithms are applied:

1. Computation of the impulse response of the car to each microphone (deconvolution algorithm).
2. Detection of the first incoming path, assuming Line of Sight, leading to time difference of arrival (TDOA) samples.
3. Computation of the location based on the detected TDOAs and assuming known (x, y) locations of the microphones.
4. (Optional) The microphone locations could be measured by hand, but can also be calculated (offline) by measuring accurately all pairwise distances between the microphones, and applying the Multidimensional Scaling (MDS) algorithm.

You have already constructed the first two algorithms as part of EE2T11 Telecommunication A practicum.

This Module will take 2 labdays. An outline of the assignments is as follows.

Audio beacon programming

Programming the audio waveform is done in Matlab using `EPO4Communications` commands (see Section 2.2.2); we can only modify a few parameters of the firmware that runs on the microcontroller in the car. The ADC audio cards receiving the microphone signals are also controlled from Matlab. All programming is done in Matlab using a PC.

Localization using TDOAs

In the EE2T11 Telecommunication A course lab you developed an algorithm to do channel estimation for 2 microphone signals both receiving a beacon (training) signal. Then you compared the estimated channel responses: The channel responses are similar, but slightly different. You focused on a time offset between the two responses which relates to the difference in distances between the transmitter and each microphone. This allowed you to compute the TDOA between two microphones.

Now, we extend this algorithm to a system with 5 microphones and a 5-channel analog-to-digital converter connected to a separate PC. This setup is used during System Integration. If you compute the TDOAs between all pairs of microphones, you will have 10 TDOA pairs. Using this information, and knowing the locations of the microphones, can you locate the beacon?

If there is time left, we can consider the following extension. So far, we assumed that the microphones have a known location. However, initially this is not the case. Instead of measuring the (x, y, z) locations of all microphones by hand with a ruler or measuring tape, we can try to do this differently. We transmit a beacon signal from the location of each microphone and measure the responses at the other microphones, converting them into TDOAs. The MDS algorithm can be used to compute the locations of the microphones from these observations.

Reporting

The results should be discussed in a section of your final report. The discussion should consist of (brief) answers to the questions posed in the text of the assignments, the requested Matlab plots, and a discussion on the results you obtained. Provide the Matlab code in an appendix.

Hints

- Split your Matlab code up into separate functions. Make separate scripts to start a test (`test.m`), which will call a script for data generation (`datagen.m`), run some functions on the data, and calls a script for showing plots (`show.m`).
- Don't write code and assume it works right away. Debug instead every function separately using simple test inputs for which you know the outcome. Then proceed to debug all functions working together.

7.2 TESTING THE AUDIO BEACON

Channel estimation algorithms

It is assumed that you have working Matlab algorithms for estimating channel responses. You developed these during EE2T11 Telecommunication A practicum.

The channel estimation problem is the following: given knowledge of a transmitted signal $x[n]$ at the loudspeaker, and a measured signal $y[n]$ at the microphone, estimate the audio channel $h[n]$ that has filtered $x[n]$ via the convolution $y[n] = x[n] * h[n]$.

To estimate $h[n]$, three alternative algorithms were described:

- Deconvolution in time domain: Involves matrix inversion. It is computationally complex and requires lots of memory (easily more than what the available PCs can handle).
- The matched filter: Avoids the matrix inversion. As it is equivalent to a convolution with a reverse signal $x[-n]$, it can be computed efficiently using the FFT. Its performance heavily depends on having the correct (i.e. accurate) "reference" or "training" signal to correlate your measurements with. Also, the signal should have good autocorrelation properties: $x[n] * x[-n]$ should be close to a delta spike. Large sidelobes will lead to confusion.
- Deconvolution in frequency domain: Involves the FFT. It computes the same channel as via deconvolution in time domain, but is much more efficient: convolution in time becomes pointwise multiplication in frequency, hence for deconvolution in frequency domain, we only need a pointwise division, followed by an IFFT to obtain the time-domain channel impulse response. But note: FFT and IFFT lead to periodicities (the result is cyclic and samples $x[n]$ at negative time reappear at the "large n" part of the signal).

  You will also need to Implement a "threshold" to set non-invertible or very small frequency coefficients of $x[n]$ to zero, which otherwise will lead to noise amplification. The performance depends on this threshold, which is data dependent and has to be chosen heuristically.

In each case, a reference signal $x[n]$ is required. To obtain this signal $x[n]$, we could rely on a model of that signal (we know the code and thus the waveform that is set to play by the loudspeaker), but it is more accurate to measure it as it leaves the loudspeaker, because we then include the loudspeaker and microphone responses as well.

Testing the audio beacon

The instructions below refer to the physical setup, but you can emulate this for the Virtual KITT with the same commands. Refer to Appendix A. You will need to initialize the microphone locations and the car position. After creating KITT, the beacon signals are recorded using

```
X = EPOCommunications('receive');
```

- Switch on the audio beacon using its standard settings, and make recordings of the transmitted sequence (1 microphone, for the moment).

  Make a clean recording at close distance to the microphone to acquire a reference signal for the channel estimation algorithm. In Matlab, strip leading and trailing small values so that the reference signal is as short as possible. Save this separately; you can use the same reference signal until you reprogram the beacon to other settings.

- Make a script to record a data segment, estimate the channel, and detect the first dominant peak.

  Test the channel estimation performance under various circumstances. Is it satisfactory? I.e., do you see a clear peak at the beginning of the estimated channel impulse response?

  Make a plot of a typical estimated channel impulse response, also indicate the estimated peak location.

  Hint: You will not be synchronized to the beacon signal; you can view this as an arbitrary unknown delay (offset) of the channel impulse response. This effect disappears when we compare two microphone channels (TDOA). However, your script must be robust for any unknown delay. E.g., it must first find the start of the actual transmission after a period of silence.

  Note that the audio beacon "beams" a lot of energy to the ceiling; the most dominant path may not be the direct path. Thus, you should try to detect the *first* significant path instead of the strongest path.

- Now make two-channel recordings and see if you can estimate the TDOA and distance between two microphones correctly (place the beacon and the two microphones on a single line, as before with the PC loudspeaker).

  Do you obtain reliable results?

Reprogramming the audio beacon

For this part, you will need to refer to the manual on programming the audio beacon in Appendix B.

In the course labs of EE2T11 Telecommunications A, you have considered suitable or "optimal" design parameters for the audio beacon. The objective was to be able to localize accurately over a distance of 6 meter, in the presence of interfering signals. We found that it is best to use a random binary code that has good autocorrelation properties (narrow "envelope" peak of the autocorrelation function, equivalent to a large bandwidth in frequency domain). The code length should be as large as possible, to maximize the energy in the code. The bit frequency parameter should probably be large, so that a large bandwidth is occupied. The carrier frequency should in theory be minimum or maximum (rather than some value in between), and match with the sample frequency. In practice, both low and very high frequencies may not be transmitted very well by the loudspeaker/microphones, so you will have to do some experiments to find out.

The code itself could be randomly generated, or you can try some optimal codes (check communication theory literature for "Gold codes").

We want a quick detection of the location, therefore the training signal should repeat itself as often as possible. Also there should be detectable gaps in the transmission (taking into account that the channel length of audio environments is quite long especially in big halls without much damping on the walls).

- Reprogramming the audio beacon is done using `EPOCommunications` commands in Matlab. Test the autocorrelation properties and channel estimation algorithm using your new settings.

- Suppose you and your neighbors use the same code. What will happen during channel estimation? Why is it necessary that both codes are "not correlated"? (In what part of your receiver algorithm does this correlation play a role?)

7.3 LOCATION ESTIMATION

The general question studied in this Module is: how can you locate the car using time-difference of arrival (TDOA) measurements made at microphones positioned at known locations? The audio beacon transmits signals which are received by 4 to 5 microphones. Depending on the distance to each microphone, the signal arrives a little bit earlier or later, and you can convert that into physical distances. For each pair of microphones, we can compute this TDOA, or the physical difference in propagation distance. If you have a large enough number of microphones (4 should work), then you can calculate the (x,y) location of the transmitter using a Least Squares algorithm.

It is assumed that you have working TDOA estimation code from the EE2T11 Telecommunications A course lab.

5-Channel TDOA measurements

During the lab day we follow the schedule published on Brightspace. The plan for today is to make initial 5-channel measurements of the beacon, so that a subgroup working on localization can start work on its algorithms. At due time, you will be called by an assistant to take your place at one of the actual set-up with 5 microphones and a separate PC in any of the designated rooms.

Important: Bring KITT. Make sure its audio beacon is programmed with your designed audio code, and make sure you write down the settings that generate it. Bring your own USB memory stick as backup in case the PC has no internet access.

- Do a test measurement of your beacon using the set-up with 5 microphones. Collect data sets at a number of (known) locations; for each location, collect several seconds of time-domain data. Also choose some sufficiently "random" locations. Store the measurement data sets (annotate them so you know what was the measurement set-up: locations of the microphones, true location of the beacon, height of the microphones and the transmitter). You can use Table 7.1 for this.

  For later use, you will also need to make recordings where the beacon is very close to each of the microphones, so that you can estimate the distance of this microphone to each of the other microphones. At the same time, the signal at the nearby microphone can be used as a reference signal in the matched filter/deconvolution. Therefore, make sure you avoid clipping at the nearby microphone.

- Save the recording (give a useful name to the mat file so that you can recognize the scenario later).

Table 7.1: Measurement table

Setting of the beacon parameters:	Nbits: Freq0: Freq1: Count3:
(x, y, z) location of the microphones:	Mic 1: Mic 2: Mic 3: Mic 4: Mic 5:
Measurement 1: Beacon at Mic 1	
Measurement 2: Beacon at Mic 2	
Measurement 3: Beacon at Mic 3	
Measurement 4: Beacon at Mic 4	
Measurement 5: Beacon at Mic 5 (ref)	
Measurement 6: Beacon in center	Location:
Measurement 7:	Location:
Measurement 8:	Location:
Measurement 9:	Location:
Measurement 10:	Location:
Measurement 11:	Location:

Multichannel measurements

During System Integration, you will need to make your own Matlab code to access the 5-channel measurement setup. We use an 8-channel ADC that connects to the PC. This interface is driven by a special driver and Matlab routines. You can find the relevant files on Brightspace. These are packaged in `PA_wav_x64.zip`, which contains files for 64-bit Matlab: `pawavplaya.dll` and `pawavplaya.mexw64`.

To record audio:

```
inputbuffer = pa_wavrecord(channels,...
                           nsamples, [samplerate],...
                           [deviceid],['asio']);
```

For example: to read from microphones 1 to 4 use

```
matrix = pa_wavrecord(1, 4, Fs*Trec, Fs, 'asio');
```

Type `help pa_wavrecord` to access the help file.

7.4 LOCALIZATION USING TDOA INFORMATION

Now that we have measured multichannel data, let's implement the localization algorithm of Appendix C, and see if it works in our situation.

In our application, we have the audio beacon on the car at a certain height above the ground (we define this as $z = 0$), but we intend to place the microphones on stands at a different height (all microphones at the same height). You can assume that the height difference between the audio beacon of the car and the microphones, h, is known.

Testing the TDOA localization algorithm

We first work with simulated Matlab data.

- Implement the given 2-D algorithm in Matlab. Test it using simulated data, assuming perfect TDOA (distance) information is available. Do you obtain the correct location of the transmitter?
- There are (at least) two sources of model mismatch.
    1. In the algorithm, it is assumed that the car and the microphones are all in the same plane. In practice there is a height difference. The microphones are on stands at about 30 cm above the car beacon. Thus, the measured (x, y, z) distances are slightly larger than the modeled (x, y) distance.

       How sensitive is the 2-D algorithm to unequal heights? (In your simulation, vary the heights by 10 cm and see by how much the estimated location varies.)

       If you notice the 2-D model is not accurate enough, then extend the algorithm to 3-D. Are 5 microphones sufficient to do the estimation in 3-D?
    2. In our application, we estimate time differences, and need to translate this to distance differences using the speed of sound, approximately 340 m/s. However, the speed of sound is not very accurately known, it depends on the temperature, humidity, air pressure, etc.

       How sensitive are your estimations to this parameter? E.g., how much does a 1% variation in speed of sound impact the distance estimate?

Applying the algorithms to the measurement data

- From the multichannel measurement data, recover first the TDOAs using your TDOA estimation algorithm.

  For each location and each microphone pair, you will have a sequence of TDOAs; convert into distance. Store the results in a table.

Estimate the mean and variance of this sequence. Compute the true distance difference and compare: are your estimates accurate? If not, perhaps you should improve the peak detection algorithm or do an outlier detection to improve the results.

- If the TDOAs are reliable, then test your location estimation algorithm on the TDOA data.

  Make a plot of the room and show the true locations and estimated locations. Are the results satisfying? What is the accuracy in practice?

Possible extensions

The provided algorithm is simple but unreliable for certain locations. What happens if the distance of the car to two microphones is equal (symmetry positions)? In that case, one column of the matrix that you try to invert is zero. During System Integration, you can search the literature and try to implement more advanced algorithms, e.g.,

- Stephen Bancroft, "An algebraic solution of the GSP equations", IEEE Transactions on Aerospace and Electronic Systems, vol.21, no.7, pp.56-59, January 1985.
- Amir Beck, Petre Stoica, and Jian Li, "Exact and Approximate Solutions of Source Localization Problems", IEEE Transactions on Signal Processing, vol.56, no.5, pp. 1770-1778, May 2008.

The literature on this topic is actually very rich. The latter paper gives a good overview. You can also try to solve for the true solution using the Matlab Nonlinear Least Squares optimization tool. However, this function may be too slow for our application.

In previous years, some students have implemented a grid search, in which the room is partitioned into a dense grid of possible positions, and each location is tested against the TDOA data to find the best fit. You could do this in two steps: first coarse, then fine.

7.5 OPTIONAL EXTRA: LOCATING THE MICROPHONES

If you have time left, this may be an interesting extension. To prepare study Appendix C.2: Tutorial on the MDS algorithm. In this tutorial, it is shown how, if you know the distances between all pairs of nodes, you can estimate their (x_i, y_i) locations, up to a translation and a rotation. You could use this algorithm to find the locations of KITT and the microphones at once. This because to measure the microphone positions by hand is time-consuming and inaccurate (especially in 3-D).

Assuming that you have 4 microphones, all at the same height h, so that you have a 2-D estimation problem. To find the distances between microphones, you place the audio beacon at the location of each microphone in turn, and measure the response at each other microphone. Since the audio beacon is not synchronized, you also measure the microphone response at the location of the beacon (be careful the microphone does not clip).

Task:

- Implement the MDS algorithm, and test it on synthetic noise-free data. Verify that you can find the correct locations (up to translation/rotation).

  To fix these unknowns, assign numbers to each microphone (1,2,3,4), assign a location $(0,0)$ to microphone 1, a location $(+x,0)$ to microphone 2, and a positive y-location to microphone 3.

- Answer a number of questions.

  Using 4 microphones, how many dimensions can you estimate? In particular, can you estimate a 3-D situation? Would that be useful in our application?

  Can the algorithm also be used to estimate the location of other objects, such as the starting position and target end position of the car, perhaps the locations of known obstacles? Note that in these cases, you don't have a microphone at these additional locations, so that you don't have distance measurements of this location to each of the microphones. But you can use the previously used location estimation algorithm.

Using the multichannel measurement data:

- From the measurement data, recover first the TDOAs.

- Test your extended MDS algorithm on the data, and estimate the locations of the N microphones. Also estimate the mean and variance of the location estimates. Compare to the true locations.

  Draw a map of the estimated locations (multiple estimates) and the true locations.

  Do you obtain satisfactory results? What is the accuracy in practice? Is that sufficient for driving the car and reaching the target? Are the results reliable (i.e., few outliers)?

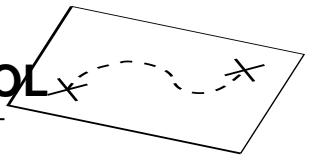
7.6 REPORTING

As part of your final report, prepare a section that discusses localization, your algorithms, and your achievements on the assignments of this module. Give (brief) answers to the questions posed in the text, the requested Matlab plots, and to-the-point discussions on the results you obtained. Provide the Matlab code in an appendix.

At this point, you should be able to locate the car using the beacon signal. During System Integration, use this as input for the trajectory tracking in a feedback loop.

Chapter **8**

MODULE 6: STATE TRACKING AND CONTROL



Contents

This section contains information that you will need when preparing for the second part: the final challenge.

The ultimate goal is to let KITT drive a certain predefined trajectory. In this final challenge you are on your own and no explicit steps are provided to help you through the process. Nevertheless, a few hints are given on how you could attack the problem (but many alternative solution approaches exist). To make life even more interesting, you can design a mix of voice control and autonomous driving.

Previously, we designed a controller to stop at a prescribed location. We will explain how the same mechanism can be used to control KITT in any predefined trajectory. The key to achieving this two-dimensional control is *decoupling*. We decouple keeping KITT on the trajectory and moving KITT along the trajectory.

Learning objectives Physical model for steering a car. Extension of control theory knowledge.

Deliverable None. This module provides ideas you may use.

Preparation Read the chapter.

Time duration Two sessions for developing the control system in Matlab and debugging it. Two or more sessions (evolving into system integration) for trajectory control analysis, development, and verification. Three homework sessions for study and report writing.

8.1 DYNAMIC MODEL FOR STEERING A CAR

This section discusses the equations that are implemented in the Virtual KITT model. This is supposed to be a good representation of steering with the real KITT (although the model has never been verified, and probably needs calibration to fit well). The same equations may inspire you to develop your control algorithms.

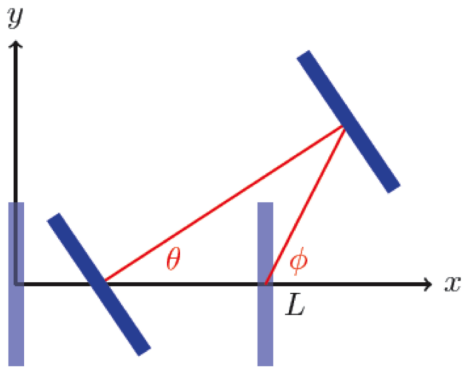


Figure 8.1: Approximation of KITT's movement

If we steer a car by making the front wheels make an angle to the baseline, then the car will follow a circle. The angle ϕ of the wheels will determine the radius R of the circle (if the angle is zero, then the radius will become infinity: driving straight is a limit case). The dynamic model for driving on a line (Module 4) remains valid, except that it now describes the distance we drive on the circle.

Thus, the hard part at this point is to derive a relation between the angle ϕ and the radius R.

Consider KITT with distance L between both wheel axes. The rear axis of the car is located at $[0,0]$ and the front axis at $[L,0]$ in a way that they are parallel to each other. The front wheels are turned an angle ϕ relative to the positive x-axis. If the car drives at a speed v, then after a very small time Δt, the rear axis is located at

$$[v\cos(\phi)\Delta t, \quad 0]$$

and the front axis at

$$[L + v\cos(\phi)\Delta t, \quad v\sin(\phi)\Delta t].$$

The situation is sketched in Figure 8.1.

The car was first parallel to the positive x-axis. Therefore, if θ denotes the angle of the car relative to the positive x-axis,

$$\theta(t) = 0$$

and

$$\theta(t + \Delta t) = \arctan\left[\frac{v\sin(\phi)\Delta t}{L}\right].$$

We can now evaluate the rate at which KITT turns by

$$\begin{aligned}\frac{d}{dt}\theta(t) &= \lim_{\Delta t\to 0}\frac{\theta(t+\Delta t)-\theta(t)}{\Delta t}\\ &= \lim_{\Delta t\to 0}\frac{\arctan\left[v\sin(\phi)\Delta t/L\right]}{\Delta t}\\ &= \lim_{\Delta t\to 0}\left[1+\frac{v^2\sin(\phi)\Delta t^2}{L^2}\right]^{-1}\frac{v\sin(\phi)}{L}\\ &= \frac{v\sin(\phi)}{L}.\end{aligned} \tag{8.1}$$

We have found a relationship between the angle of KITT's wheels and the rate at which KITT turns.

Tasks:

- Assume a constant velocity. Derive an equation for the radius R of the circle. Show the derivation in your report.
- Is this equation also valid in case the velocity is not constant?
- For Virtual KITT, the maximal angle of the wheels is set at ± 0.45 rad. The wheelbase L is 33.5 cm (see Figure 2.1). What is R? Verify this by running the Virtual KITT model (in 'public' mode) and measuring R in your plot.

  Try to estimate these parameters for the real KITT as well.
- Make a more detailed plot than Figure 8.1, showing also the center of the circle, C. Which part of the car (back wheels, center, front wheels) actually rotates around C?
- Define a unit direction vector $\mathbf{d}_0$, and a vector $\mathbf{d}_0^{\perp}$ orthogonal to it, and derive the new direction vector $\mathbf{d}_1$ and the vector $\mathbf{d}_1^{\perp}$ orthogonal to it. From this, compute the new position of the car $\mathbf{x}_1$ if initially it was at position $\mathbf{x}_0$.

  Hint: You may want to use a rotation matrix

$$\mathbf{R}(\theta) = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}.$$

8.2 CONTROL

8.2.1 Getting to your goal

Assume that we are able to keep KITT on any trajectory. Obviously, we have to slow down when approaching our target position. However, using bang-bang control, it is not clear how we should use our curves to find the moment at which KITT should start slowing down. Bang-bang control was used

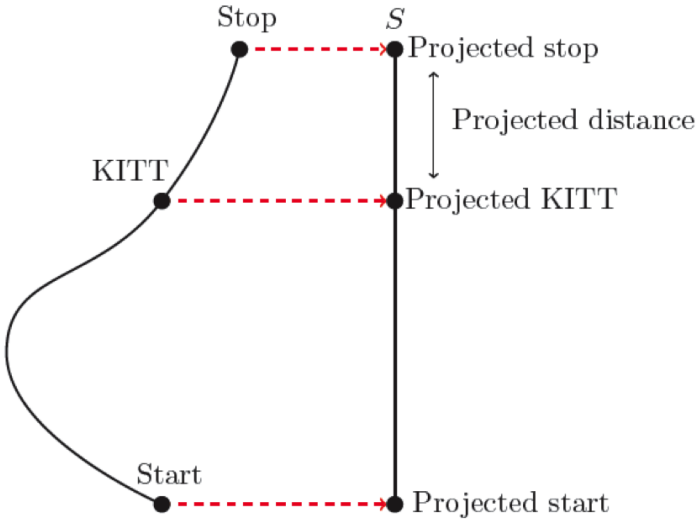


Figure 8.2: KITT's trajectory

to control KITT in a one-dimensional space. Following a trajectory requires control in a plane, which is a two-dimensional space. A solution to this problem is given by projection. If we could project our two-dimensional space onto a one-dimensional space S, then we could use KITT's position in S for one-dimensional control. Figure 8.2 depicts this idea.

Let KITT's trajectory be given by

$$\mathbf{x}(t) = [x(t), y(t)].$$

We define the projection of (x, y) onto S by

$$z = z(x, y)$$

where $z \in S$. The projected position z can be regarded as the distance remaining to the wall. KITT's speed should then be continuously controlled so that z approaches zero without overshooting. Thus using z as a measure for distance allows for one-dimensional control in a plane. However, we should choose our projection very carefully such that all assumptions remain valid. To this end, the projected position z should behave like the distance remaining to the wall. Therefore, if KITT approaches its target position, this should be reflected by z. More specifically, the distance which KITT moves along the trajectory should be reflected by a decrease in z. This requirement is satisfied if KITT's speed is conserved by the projection. We therefore state

$$-\frac{d}{dt}z = \|\mathbf{x}'(t)\|_2.$$

We can now find an explicit expression for z by integrating with respect to time. Integrating from a time

τ to infinity yields

$$-\int_{\tau}^{\infty} \frac{d}{dt} z dt = z(\mathbf{x}(\tau)) - z(\mathbf{x}(\infty)) = \int_{\tau}^{\infty} \|\mathbf{x}'(t)\|_2 dt$$

The end of our trajectory $\mathbf{x}(\infty)$ should correspond to KITT who has stopped just before the wall. Therefore

$$z(\mathbf{x}(\infty)) = 0.$$

This yields an explicit expression for z given by

$$z(\mathbf{x}(t)) = \int_{t}^{\infty} \|\mathbf{x}'(\tau)\|_2 d\tau. \tag{8.2}$$

Note that we have replaced the integration variable by τ. Inspection of Equation 8.2 yields that the projection is given by arc length of the curve from KITT's current position to its target position. However, evalution of this integral requires knowledge of KITT's position in the future, which we do not have. It is clear that one-dimensional control in a plane requires an *estimation* of KITT's future movement. This estimation is exactly given by the planned trajectory. We can now carefully state our wanted projection. The projected position z is given by the arc length of the planned trajectory from KITT's current position to its target position. This allows for bang-bang control along any trajectory in a plane.

8.2.2 Following your goal

In the previous section we assumed that we were able to let KITT follow any predefined trajectory. We will now design a controller which keeps KITT on track.

Suppose KITT is driving on its trajectory S, where its angle with respect to the x-axis is given by θ. Let the angle tangent to its trajectory S be given by θ_t. Ideally, this angle should be equal to KITT's angle. If KITT's angle deviates from this angle, KITT should turn its wheels to steer back. This motivates a feedback law given by

$$\phi = -k(\theta - \theta_t)$$

for a positive k. Substituting in Equation 8.1 yields the autonomous non-linear system

$$L\frac{d}{dt}\theta + v\sin(k(\theta - \theta_t)) = 0. \tag{8.3}$$

We should choose k such that this system is asymptotically stable. To investigate the stability, we introduce a potential function given by

$$T(\theta) = \frac{1}{2}(\theta - \theta_t)^2.$$

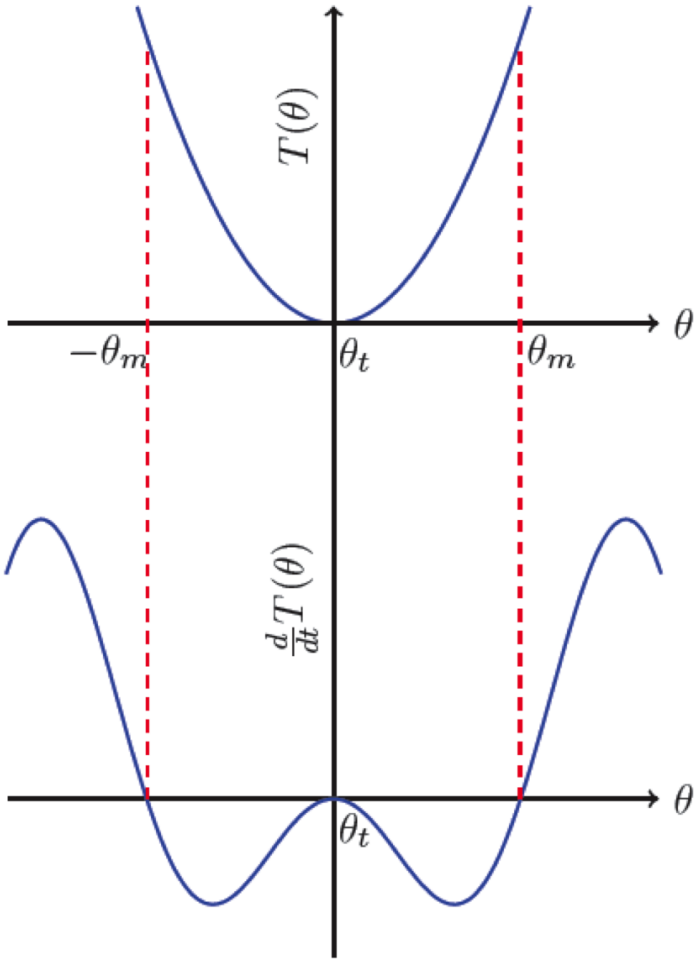


Figure 8.3: Graph of the potential function and its derivative

A first observation is that $T(\theta) = 0$ if and only if $\theta = \theta_t$, which is exactly what we want. Second, notice that $T(\theta) \geq 0$. We can conclude that T's minimum corresponds to our equilibrium point. Notice that

$$\frac{d}{dt}T(\theta) = -(\theta - \theta_t)\frac{v\sin(k(\theta - \theta_t))}{L}. \tag{8.4}$$

Figure 8.3 depicts both $T(\theta)$ and its time-derivative.

Consider KITT's angle at any instant. If $|\theta| < \theta_m$, then by Figure 8.3 the potential function will have a negative derivative. But then it will lower value in time. Thus T will keep decreasing until $\theta = \theta_t$. So in conclusion, if $|\theta| < \theta_m$, then θ will converge to its equilibrium point. By inspection of Equation 8.4 we

can now state that Equation 8.3 is locally asymptotically stable for any

$$-\frac{\pi}{k} < \theta - \theta_t < \frac{\pi}{k}.$$

This treatment is also called investigation of *Lyapunov stability*, where T is called the *Lyapunov function*. It is an extensive topic in the control literature.

8.2.3 State tracking and state error feedback

In the signal processing/control literature, Kalman filters are used to estimate the state (position, velocity) of a system, given its inputs and observed (noisy) outputs. This assumes linear systems in state-space form (either continuous-time or discrete-time): in discrete time, we model

$$\begin{cases} \mathbf{x}_k &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{v}_k \end{cases}$$

Here, $\mathbf{u}_k$ is the system input at time k, $\mathbf{x}$ is the (true) state vector, and $\mathbf{y}_k$ is an observation. For example, for moving in 1D on a line, without friction,

$$\mathbf{x} = \begin{bmatrix} x \\ v \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{u} = F_a\,, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}, \qquad \mathbf{C} = [1 \quad 0]\,.$$

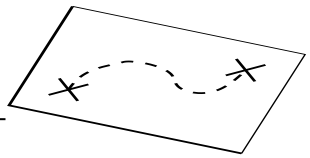
The disturbances $\mathbf{v}_k$ and $\mathbf{w}_k$ are unknown additive noise processes at the input and the output, respectively, that will make the state and the output deviate from their real values. The Kalman filter is an estimator for $\mathbf{x}_k$ given $\mathbf{u}_k$ and $\mathbf{y}_k$ (and the previous state estimate at time $k-1$; it tries to track this state); it is a topic for MSc-level courses (e.g., EE4C03).

Unfortunately, steering a car is highly nonlinear. We could try to use an Extended Kalman filter. Alternatively, if you have implemented your own car model, you can use that to predict the next location and state, given the same inputs, and use the observed error in position to update/correct your state estimate, similar to what a Kalman filter does.

A complete theory is not developed here, the above is meant as inspiration for your own solution!

Chapter 9

FINAL CHALLENGE



Contents

You have made it to the final part: System Integration. At this moment you have all the separate subsystems working. However, this does not mean the project is completed. The most important part of the project is yet to come: making it work all together.

You and your team have just a few weeks to achieve that. The project is concluded by a final challenge where you demonstrate your achievements in competition to other teams. You also have to prepare a final report and a short presentation. The report is discussed after the oral presentation.

Demonstration At the date of the final challenge, show the capabilities of KITT regarding autonomous driving. The demonstration is done as a group. Your performance contributes to the final grade.

Report A final report explaining your design with its technical details. This must contain the design choices of your complete design, the expected capabilities of the resulting system, the problems you faced with linking the parts together and a brief summary of the previous modules comprising the relevant conclusions. Suggested length: 30 to at most 40 pages.

The report is prepared as a team. The report is graded and contributes to your final grade.

Presentation and discussion A presentation of 5 min, followed by an interview of approximately 25 min in which you explain your design in front of an examination committee. This contributes to your final grade.

Preparation Completed all preceding modules.

Time duration Three lab sessions for system integration and testing, one lab session for the demonstration, five homework sessions for report writing and presentation preparation, 45 mins for the presentation and discussion.

9.1 SYSTEM INTEGRATION

Before you can start with the integration part, it is necessary to have completed all the previous subsystems. If this is not the case, make sure this does not affect your final design and contact the consultants for advice.

To link all the parts together, you must communicate with the complete group about the responsibilities and define precisely what in- and outputs are generated by each subsystem.

Design requirements

The assignment during system integration is to complete the big final design such that you can compete succesfully. The design must contain:

(i) A positioning system;

(ii) a control system that controls the complete car and drives the car to certain given points in the arena;

(iii) optionally: a voice recognition system;

(iv) optionally: an obstacle-avoidance system.

Work on the obstacle-avoidance aspect only after completing first the mandatory requirements.

The constraints to the design are given by the tasks KITT must fulfill. These are described in Section 9.2. General points are:

- During setup, you must be able to enter the coordinates of the target and at least one waypoint.
- When reaching a waypoint, KITT halts for at least 10 s such that its distance to the waypoint can be measured.
- During the race, KITT or the keyboard cannot be touched. After KITT starts to drive, everything is autonomous or by voice control.
- The initial position of the car is known; also its orientation is known.
- Targets should be reached with an accuracy of at least 30 cm.
- The system presents on the PC a plot which visualizes the past and current positions of KITT.
- Bonus points are given to the fastest team.

The main objective is to drive to the target, possibly via a waypoint. Optional tasks are:

- The field contains obstacles, to be detected by the anticollision system. After detection, the system takes an evasive action to reach the desired target or waypoint.
- A second KITT is in the field with its audio beacon on. This might affect your localization accuracy.
- A Free Challenge: impress us with your own challenge!

Bonus points are given for completing the optional features.

For the control system, you will have to study Section 8.2 for a system model of the car including steering. Not all background for controlling this system is given in that section, and you will have to search the literature for solutions, or invent something yourself (in the past, we have seen solutions where cars also drive backwards to correct route, avoid obstacles or prevent going out the field). The control loop is probably rather different than the one used in the mid-term competition: now you can essentially drive at constant speed, but the control of a steering angle makes this a nonlinear problem.

Speed is not a requirement in the final challenge. By driving faster, you may reach the target faster, but trajectory tracking becomes more difficult, location updates must be computed faster, and you have less time to avoid obstacles.

9.2 DEMONSTRATION

The final challenge for EPO4 consists of 4 challenges and a "free challenge". In order to get a passing grade, you have to successfully pass at least the first challenge. With every other challenge you pass you will get a higher grade. You will have maximum two attempts for each challenge. If you fail a challenge, you may not compete in further challenges, except for the "free challenge".

Before each challenge you are allowed to measure the field and the position of the destination (and waypoint). A challenge starts at a given starting position. The orientation of the car is always 90° with respect to one side of the field. Once the start command has been issued you may not touch KITT nor the PC except for an emergency stop.

KITT may stop everywhere and as many times as needed. There is no time limit except for the fact that everything, including preparation and cleaning up, should happen within 30 minutes (or less). Needless to say, you must use the audio beacon for localization, and the ultrasonic sensors for obstacle detection (no "open loop" solutions allowed).

If you implement voice commands, you may use these to override the autonomous behavior of the car, i.e., as an emergency action. Suggested commands are Setup, Go, Stop, Slow, Left, Right, Back. You can invent your own "language" for these commands. You are allowed to give at most 10 voice commands per Challenge.

Challenge A (65 points)

- KITT drives from the starting position to a specific point A in the field.

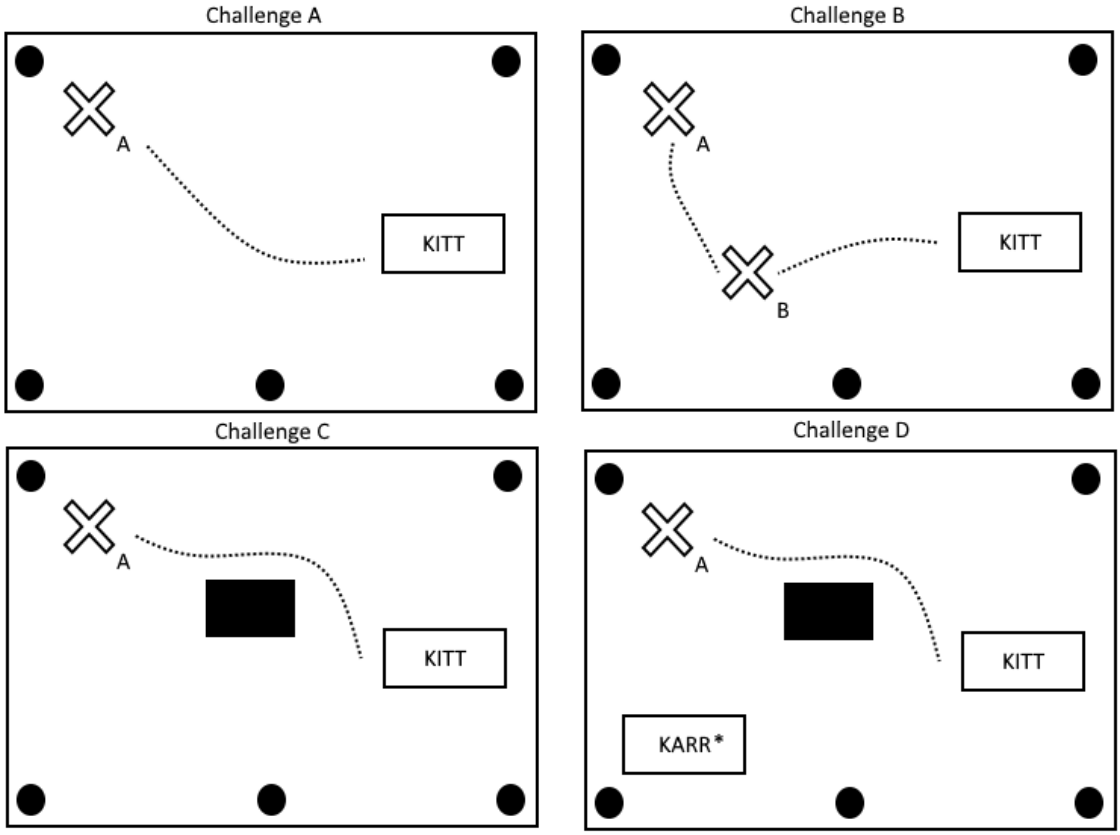


Figure 9.1: Overview of the Challenges. (*In the old television series, KARR is the archenemy of KITT)

- Once KITT reaches the destination it must stop and the PC must give a signal.
- There are no obstacles on route.
- There are no other cars involved.

Challenge B (10 points)

- KITT drives from the starting position to point A via another point B in the field.
- When KITT reaches the waypoint (B) you must halt for 10 s such that the examinator has time to measure the distance to the waypoint
- When KITT reaches the destination (A) it must stop and the PC must give a signal.
- There are no obstacles on route.
- There are no other cars involved.

Challenge C (10 points)

- KITT drives from the starting position to a specific point A in the field.
- When KITT reaches the destination (A) it must stop and the PC must give a signal.
- There is an obstacle on route (two paper bins on top of each other) that has to be avoided.
- When KITT finds the obstacle, its position must appear in the GUI.
- There are no other cars involved.

Challenge D (8 points)

- KITT drives from the starting position to a specific point A in the field.
- When KITT reaches the destination it must stop and the PC must give a signal.
- There is another car involved (stationary but with working beacon), this car has to be avoided.

Free Challenge (7 points)

- Invent your own challenge and impress us! E.g. drive from A to B to C where you will need to drive backwards to make the turn. Or drive the entire route backwards.
- Points awarded depending on the difficulty and creativity of the task.

Grading

– If you complete each task perfectly you receive the total amount of points. Penalty points are deducted for missing the target or hitting an obstacle. The grading for a task cannot be negative. You fail a challenge (and therefore cannot continue with the next challenges) if you either miss the target by more than 60 cm or you lose all the points the challenge can give you.

– The measurements are done with respect to the center of the loudspeaker on top of KITT.

– In *Challenge A* you may miss the destination by 30 cm but for every additional 10 cm you will lose 5 points. You can't lose more than 15 points.

– In *Challenges B–D* you may miss the targets by 30 cm but for every additional 10 cm you will lose 2.5 points, this holds for both the way point as the destination.

– For each time KITT hits a obstacle or another car you will lose 2.5 points.

Bonus The time you take to complete the Challenges will be measured. The fastest team will receive 10 bonus points and the second fastest will receive 5 bonus points. (It is not possible to receive more than 100 points for the whole competition.)

9.3 FINAL REPORT

Instructions for the final report are similar to those of the midterm report (see section 6.2). Aim for a **well-structured**, compact yet complete report of about 30 pages (plus Matlab code in an appendix). Do not forget to report on testing/verification: how do you test (each subsystem, and the entire system), what are the results from the test, what do you conclude. Include the results of the final challenge in the report as well.

The report is judged by committee members that are not indepth familiar with EPO4, or the manual. Your report has to be sufficiently self-contained.

The report is separate from the mid-term report. Only include what you need for your final challenge.

The submission deadline is listed in Brightspace, typically one day after the final challenge. Submit your report using the corresponding submission folder.

Refer to the external document (included in an announcement in Brightspace) for a detailed description of the evaluation based on rubrics.

9.4 FINAL PRESENTATION AND DISCUSSION

In week 9 (consult Brightspace for the exact date), you present and defend your final report in front of an examination committee. The examinators will ask questions about your design choices and aspects of team work. This will be part of your grade.a

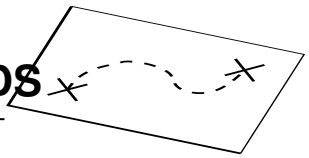
The presentation lasts at most 5 min. This is too short to have all team members presenting. Focus on the highlights and special features of the design, and mention the work breakdown and distribution of tasks to team members.

The examination will last about 30 min. After the examination you will be asked to fill in a peer review form. Individual grades are differentiated depending on staff observations and the outcome of the peer review.

Refer to the external document (included in an announcement in Brightspace) for a detailed description of the evaluation based on rubrics.

Appendix ***A***

VIRTUAL KITT COMMUNICATION COMMANDS



Contents

During Corona times, a Virtual KITT Matlab model has been developed that mimics the real KITT. It uses the same `EPOCommunications` command interface to control the car, so it would be easy to switch from the model to the real car, and back: great for debugging your code!

Virtual KITT is implemented as a Matlab `classdef` object `car`. Each microphone is a Matlab object `mic`, the microphone array is an object `micarray`, and the beacon is an object `beacon`. While you can inspect the code, and have access to some of their properties and methods, you don't have to know any details as this is handled by the `EPOCommunications()` function, as described in this Appendix.

The main catch on the model is: it has not been calibrated, meaning its behavior might be different than that of the real car. However, you could try to tweak the internal parameters to obtain a better fit. (See `car_accel.m`.)

To run the model, look for a file `carmodel.zip` on Brightspace, and install the files in a directory `carmodel`. You may have to adjust your Matlab path so Matlab can find the files: e.g.,

```
addpath carmodel -begin
```

An included file `test_epo.m` contains a script that tests most of the functionality, and allows to drive the car using a simple user interface with WASD interaction. This also shows you how to use the most important functions of the `EPOCommunications()` interface.

A.1 EPOCOMMUNICATIONS

For Virtual KITT, the functionality of the regular EPOCommunications command is emulated by a function in a file `EPOCommunications.m`, which you must place in a directory where Matlab can find it. Of course, since the name is the same, you also have to find a way to switch between this virtual interface and the real one!

`EPOCommunications('help')` will print a bit of help on the various options.

A.2 INITIALIZATIONS

A difference with the real KITT is that, before we can use Virtual KITT, we need to define some initial values such as the initial position and direction of the car, the dimension of the arena, the positions of the microphones, etc.

Initializing the position and orientation of the car In the virtual implementation, the position is a 2D column vector $[x;y]$ specified in centimeters, and the direction is a unit-norm column vector, e.g., $[0;1]$. The position is considered to refer to the center of the car (in fact, the beacon loudspeaker). The height is assumed to be 0 and is not specified. Use

```
EPOCommunications('init','X[100;0]'); % initial position of car
EPOCommunications('init','D[0;1]'); % initial direction of car
```

If you have the position in a variable, you need to convert that to a string, e.g.,

```
x = [100;0];
EPOCommunications('init',['X' mat2str(x)]); % initial position of car
```

Initializing the microphone array During initialization, we need to specify the locations of the microphones. These locations are in 3D (also the height is specified). Typically you specify 4 or 5 microphones. First define a matrix `P` where each column is a location, e.g.

```
P = [
0 600 0 600;
0 0 600 600;
30 30 30 30]; % positions of mics
```

This defines 4 microphones at the corners of a 600×600 cm square, at a height of 30 cm. Next, we send an initialization command 'P' followed by the positions (as a string), i.e.,

```
EPOCommunications('init',['P' mat2str(P)]); % initialize positions of mics
```

We also must set the sample rate of the microphones,

```
EPOCommunications('init','J40000'); % set sample rate of mics (Hz)
```

This is coupled to the transmission rate of the beacon so that they match. Finally, we can already specify here how many samples a microphone 'receive' command (see later) will return:

```
EPOCommunications('init','N2000'); % set number of samples to return
```

Initializing the arena and defining obstacles KITT has distance sensors, but they can only be used if Virtual KITT knows where are the obstacles. So we need to define this during initialization.

Obstacles are defined as polygons, or Matlab `polyshape` objects. This is rather versatile. We need to specify the corners (vertices) of objects. There can be multiple objects, they are separated by a vector consisting of `NaN`.

```
O1 = [
200 200 260 230;
200 240 240 200]; % first obstacle
NAN = [NaN; NaN]; % spacer
O2 = [
350 380 410 410;
200 240 240 200]; % second obstacle
oo = mat2str([O1 NAN O2]);

EPOCommunications('init',['O' oo]);
```

Every column of the matrices `O1` and `O2` specifies the coordinates of a vertex of the polygon. A polygon consists of 3 or more vertices. You could create the corresponding polygon representation yourself using `polygon([O1 NAN O2]')`.

If an object overlaps another object, this creates a hole. So the arena can be defined by specifying a very large box, and inserting inside it a smaller box of the size of the actual arena—this will create a hole of the right size:

```
B1 = [
-50 650 650 -50;
-50 -50 650 650]; % outer arena boundary (outside bounding box)
NAN = [NaN; NaN]; % spacer between objects
B2 = [
-21 621 621 -21;
-21 -21 621 621]; % inner arena boundary (hole since it overlaps B1)
bb = mat2str([B1 NAN B2]);

EPOCommunications('init',['O' bb]); % initialize arena boundary
```

As is seen, the `O` command can be called several times to add additional objects to the list of obstacles.

Initializing waypoints During the race, you will be asked to verify your distance to certain waypoints (or at least one final point). They are defined in 2D as follows:

```
W = [
100 300;
100 300]; % define two waypoints
```

Each column defines a separate point. Next, initialize them using the 'W' command:

```
EPOCommunications('init',['W' mat2str(W)]); % initialize waypoints
```

Creating KITT With a regular KITT, we would issue an 'open' command to start using the Bluetooth connection over a serial port. For Virtual KITT, we issue a similar command, but now it will create a Matlab KITT object. The command will return the handle (or pointer) to this object. The regular command is

```
kitt = EPOCommunications('open') % create kitt! (hidden state)
```

The initialization values (specified with 'init' above) are used. The return value `kitt` is a handle and you can use it to inspect a lot (but not all) of the internal parameters of KITT.

There is a similar command,

```
kitt = EPOCommunications('open','P') % create kitt! now in public mode
```

This creates a 'public' version of KITT that gives access to state variables that are normally hidden (position, orientation, velocity). You can use this as training mode to develop your code.

Destroying KITT After you are completely done, you can 'close' the connection to KITT using

```
status = EPOCommunications('close') % destroy kitt :(
```

This will make the `kitt` handle invalid.

The returned `status` is a structure with fields `status.x` (position), `status.d` (direction vector), `status.v` (velocity), `status.obstacles` (current list of obstacles), and `status.crash` (true/false).

Calling any `EPOCommunications('init',...)` command will first issue a 'close'. Thus, it is not possible to first create KITT and then set its initial position.

A.3 STATUS COMMANDS

Requesting the car status The current status of the car is requested using

```
EPOCommunications('transmit','S'); % request status string
```

The status string reports the current drive commands, the ultrasonic sensor distance (cm), the battery voltage (V), and the audio status (on/off) and control parameters (code word, carrier-frequency, bit-frequency, repetition frequency). The output structure of this command is:

```
*************************
* Audio Beacon: on
* c: 0x00000000
* f_c: xxxxx
* f_b: xxxxx
* c_r: xxx
*************************
* PWM:
* Dir. xxx
* Mot. xxx
*************************
* Sensors:
* Dist. L xxx R xxx
* V_batt xx.x V
*************************
```

A sensor distance of `999` means overloading (i.e., out of range).

This 'string' version of the status is what would be received over Bluetooth from the real KITT, but it is actually rather inconvenient (it needs to be parsed). The Virtual KITT has a direct way: simply type `kitt` (i.e., the handle returned during creation),

```
>> kitt
kitt =
Version: 13 May 2020
Dynamic:
  position: [194.655;198.782] cm
  orientation: [0.84775;0.530397]
  velocity: 0 cm/s
  acceleration force: 0 N
  angle: -0.198 rad
  time since last update: 4130.58 s
Beacon:
  button: off
  Freq0: 15000 Hz
```

```
  Freq1: 5000 Hz
  Count3: 2500, repeat freq: 2 Hz
  code: 00000000
  Fs: 20000 Hz
  message: (10000 samples)
Micarray:
  mic #1 location: [0;0;30] cm, sample rate: 20000 Hz
  mic #2 location: [600;0;30] cm, sample rate: 20000 Hz
  mic #3 location: [0;600;30] cm, sample rate: 20000 Hz
  mic #4 location: [600;600;30] cm, sample rate: 20000 Hz
  beacon is off
  stored message: (10000 samples)
  received signal: 2000 samples
Sensors:
  distL = 10 cm
  distR = 10 cm
  offsetL = [20;15] cm
  offsetR = [20;-15] cm
  Minrange = 10 cm
  Maxrange = 300 cm
  Aperture = 0.523599 rad
  Obstacles:
  polyshape with properties:

     Vertices: [19x2 double]
    NumRegions: 3
     NumHoles: 1
```

The precise form of this output may vary, depending on the version of the software and whether you opened in public or hidden mode.

Object-oriented status queries As an unofficial part of the interface, the current object-oriented implementation allows direct access to various parameters, such as

```
kitt.position  % only if 'public'; cm
kitt.orientation % only if 'public'; unit-norm vector
kitt.velocity  % only if 'public'; cm/s
kitt.force  % the latest 'motor' command
kitt.angle  % the latest 'direction' command
```

```
kitt.Beacon.button % on/off switch of the beacon
kitt.Micarray.mics(1).location % mic position; cm
kitt.obstacles  % list of obstacles (polyshape array)
```

The distance sensors are queried by

```
kitt.distL  % left parking sensor; cm
kitt.distR  % right parking sensor; cm
```

Like for the real KITT, there is a noticeable response time (due to the 'Bluetooth' communication delay). Unlike the real KITT, the distance information is based on the current position, and not on the position some time ago.

To check if the current car position is within the boundary of an obstacle, do

```
tf = carcrash(kitt) % true or false
```

If a car crash is detected, the car is stopped as well.

A.4 DRIVING THE CAR

The car is told to drive in a certain direction using two commands

```
EPOCommunications('transmit','D150'); % direction command
EPOCommunications('transmit','M150'); % motor speed command
```

More generally,

```
d = int2str(direction);
f = int2str(force);
signal = ['D',d];
EPOCommunications('transmit',signal);
signal = ['M',f];
EPOCommunications('transmit',signal);
```

The 'D' command sets the direction. A value of `150` means neutral. The minimum value is `100`, which sets the wheels at an angle -0.45 rad compared to the forward direction, steering to the right. The maximum is `200`, which sets the wheels at an angle of +0.45 rad, steering to the left. Only integer values are accepted.

The 'M' command sets the acceleration force. A value of `150` is neutral (but if the car has any speed, it will continue to roll until friction will stop it). Between `151-165` means to drive forward, and between `135-149` means to drive backwards. The maximum setting corresponds to a force of 10 N and will allow KITT to reach a speed of about 200 cm/s after a few seconds.

Both commands return the current status.

For Virtual KITT, you could if you want bypass `EPOCommunications()` and set the accelerator force and steering angle directly using

```
kitt.force = 135;
kitt.angle = 150;
```

A.5 BEACON COMMANDS

Handling of the audio beacon The audio beacon is switched on or off using

```
EPOCommunications('transmit', 'A0'); % switch off audio beacon
EPOCommunications('transmit', 'A1'); % switch on audio beacon
```

The beacon signal is similar to what was used in EE2T11 Telecommunication A practicum, except that now it is possible to use an arbitrary carrier frequency, bit frequency, and repetition count. The code length is 32 bits (fixed). The code itself is arbitrary. The following four commands are available to configure the behavior of the beacon (using example values):

```
EPOCommunications('transmit', 'B5000'); % set the bit frequency
EPOCommunications('transmit', 'F15000');% set the carrier frequency
EPOCommunications('transmit', 'R2500'); % set the repetition count
EPOCommunications('transmit', 'C0xaa55aa55'); % set the audio code
```

The bit frequency determines the bandwidth used by the signal; the default value is 5 kHz.

The repetition frequency is specified by setting the repetition count. The relation between these two units is:

$$repetition\_count = bit\_frequency / repetition\_frequency$$

The repetition count is the number of bits the beacon waits before transmitting the code again. The minimum value is 32 (otherwise a new code is transmitted before the previous one finished). With the default value of the bit frequency (5 kHz) the shown (default) value of 2500 for the repetition count relates to a repetition frequency of 2 Hz (rather slow).

The fourth command is used to set the 32 bits code pattern. It is required to specify the code in C hex format (here as example `0xaa55aa55`; probably not a very good code). The default code is `0x00000000` which means no code is sent.

For Virtual KITT,

```
x = kitt.Beacon.message;
```

returns the current beacon transmit message (including trailing zeros), corresponding to the latest setting of the beacon parameters; the sample rate is derived from the microphone setting.

Accessing the microphone array To receive data from the virtual microphones, use

```
X = EPOCommunications('receive'); % receive micarray signals (matrix)
```

Here, `X` will be a matrix of size $N \times M$, where N is the number of samples specified with the 'N' command during initialization, and M is the number of microphones, as defined using the 'P' command during initialization.

You can also write

```
X = EPOCommunications('receive','N2000');
```

which will collect $N = 2000$ samples (overwriting the value set during initialization).

The microphone signals naturally have an arbitrary (but common) time offset. There is also a noticeable response time (due to the 'Bluetooth' communication delay).

In 'public' mode, the microphone signals are almost noise-free, allowing you to debug your code a bit easier. However, the performance is then not representative of running the real KITT!

Checking the waypoint distance During the race, you will be asked to drive to certain points in the arena, and to demonstrate how close you got, you will need to check the distance of KITT to the next waypoint:

```
dist = EPOCommunications('transmit','W')
```

This does two things: it reports the distance to the first waypoint in the list, but also removes the waypoint from the list. Hence, it is possible to check this distance only once!

You could if you want bypass `EPOCommunications()` and request the distance check using

```
dist = kitt.distW
```

Note that in any case, it is necessary to come to a standstill to enable a correct measurement. (Standstill is interpreted as velocity smaller than 1 cm/s.)

A.6 PLOT COMMANDS

The virtual implementation also provides some plot commands: e.g., to show the car at its current position and orientation (in 'public' mode), use

```
hand = plot(kitt,aa); % draw outline of the car (if public)
```

where `aa` is the handle to a figure axes where you want the plot to show (if you omit it, it will default to `gca`), and `hand` is a handle that you can use to remove the outline later on:

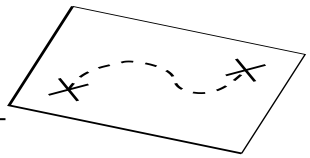
```
delete(hand)
```

If during plotting, a car crash is detected, a large '$*$' is plotted at the car position, and the car is stopped. Similarly, you can plot the obstacles, microphones and waypoints as they are known to KITT, using

```
plot(kitt.Sensors,aa); % plot obstacles and arena bounding box
plot(kitt.Micarray,aa); % plot mics as "x"
plot(kitt.Waypoints,aa); % plot waypoints as "o"
```

Appendix ***B***

PROGRAMMING THE AUDIO BEACON



Contents

This appendix describes the audio beacon signal parameters. The signal is generated by the M0 microcontroller of the LPC4357 ARM, an amplifier circuit (LM4752) and a loudspeaker. The microcontroller runs a dedicated program that produces continuously this signal on one of its PWM output pins which is amplified by an op-amp circuit and made audible by the loudspeaker.

The parameters that define the signal are described in Section B.1. These parameters can be set through the `EPOCommunications` command interface.

B.1 AUDIO BEACON SIGNAL PARAMETERS

The beacon signal is similar to what is described in EE2T11 Telecommunications A practicum, except that now it is possible to use an arbitrary carrier frequency, bit frequency, and repetition count.

The audio beacon transmits a binary code sequence using "on-off keying" (OOK). If a bit in the sequence is 0, nothing is transmitted; if the bit is 1, a modulation carrier frequency is transmitted during a certain period. The number of bits is fixed at 32.

Besides the actual bit sequence (a 32 bit code word), the parameters that determine the signal are:

- Modulation carrier frequency (parameter `Freq0` specified in Hz), at most 30 kHz, although this is probably beyond the specs of the loudspeaker and microphones;
- Bit frequency (parameter `Freq1` specified in Hz), this defines the rate at which the modulation carrier signal is switched on or off by the bits in the code word, i.e., determines the duration of a single bit (and indirectly the bandwidth of the generated signal);
- Repetition count of the bit sequence (parameter `Count3`, an integer), this specifies the number of bits the beacon waits before transmitting the code again. The minimum value is 32 (otherwise a new code is transmitted before the previous one finished). The resulting repetition frequency is

$$repetition\_frequency = bit\_frequency / repetition\_count$$

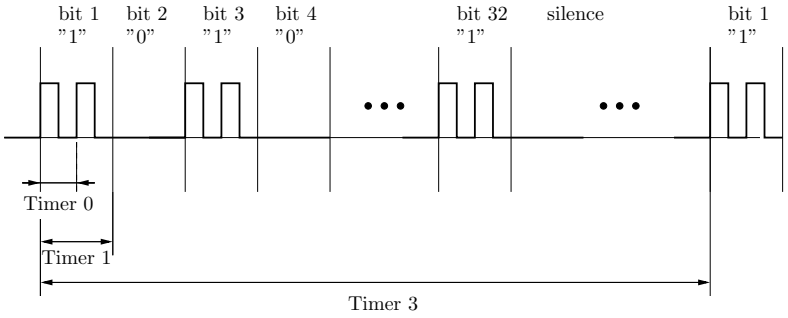


Figure B.1: An example of the pulses generated by the audio beacon.

For example, with the default value of the bit frequency (5 kHz), a repetition count of 2500 gives a repetition frequency of 2 Hz. You will probably want to have a higher repetition frequency.

These parameters are set using `EPOCommunications` commands, which sends them to the microcontroller on the car over the Bluetooth interface. The commands are described in Section 2.2.2.

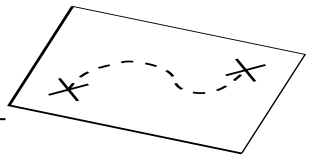
The Matlab model `refsignal` that is used in EE2T11 Telecommunications A practicum has similar parameters, but limited to a specific set of values. You can use `refsignal_new.m`, found on Brightspace.

The default setting of the audio beacon is a 32 bit code sequence bit-stream with:

Freq0	Carrier frequency	=	15000	Hz
Freq1	Bit frequency	=	5000	Hz
Count3	Repeat counter	=	64	samples
	Code word	=	`0x92340f0f`	(hex)

Appendix **C**

TDOA LOCALIZATION



Contents

C.1 TIME-DIFFERENCE OF ARRIVAL (TDOA) ALGORITHM

Let $\mathbf{x} = [x, y]^T$ be the location of the car, and call $\mathbf{x}_i = [x_i, y_i]^T, i = 1, \cdots, N$ the known locations of the microphones. The Time Difference of Arrival (TDOA) for each sensor pair (i, j) is translated to a range difference (by multiplying by the speed of sound) r_{ij}, where

$$r_{ij} = d_i - d_j\,, \qquad d_i = \|\mathbf{x} - \mathbf{x}_i\|\,, \qquad d_j = \|\mathbf{x} - \mathbf{x}_j\|$$

The objective is to compute from the available measurements, $\{r_{ij};\, i, j = 1, \cdots, N, i < j\}$ the location $\mathbf{x}$ of the car. Each range measurement specifies a hyperbolic curve in the 2-D plane of possible locations $\mathbf{x}$, and the combination of measurements asks for the intersection of the curves (this is called *multilateration*). Equivalently, we have to solve a system of quadratic equations.

Fortunately, it is possible to transform these into a system of *linear* equations, augmented with some additional "nuisance" parameters, as follows. Write

$$r_{ij} = d_i - d_j \quad \Rightarrow \quad (r_{ij} + d_j)^2 = d_i^2 \quad \Leftrightarrow \quad r_{ij}^2 + d_j^2 + 2 r_{ij} d_j = d_i^2 \tag{C.1}$$

We will use

$$\begin{array}{rrcl} & d_i^2 & = & \|\mathbf{x} - \mathbf{x}_i\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\mathbf{x} \\ \text{and} & d_j^2 & = & \|\mathbf{x} - \mathbf{x}_j\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_j^T\mathbf{x} \end{array}$$

Inserting this into (C.1), we see that the quadratic term $\|\mathbf{x}\|^2$ is eliminated:

$$\begin{array}{rcl} r_{ij}^2 + d_j^2 \;+\; 2 r_{ij} d_j = d_i^2 & & \\ \Leftrightarrow \quad r_{ij}^2 + \|\mathbf{x}\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_j^T\mathbf{x} + 2 r_{ij} d_j & & \\ \qquad\qquad = \|\mathbf{x}\|^2 + \|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\mathbf{x} & & \\ \Leftrightarrow \quad r_{ij}^2 - \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 = 2(\mathbf{x}_j - \mathbf{x}_i)^T\mathbf{x} - 2 r_{ij} d_j & & \end{array}$$

This can be written in vector notation as

$$[2(\mathbf{x}_j - \mathbf{x}_i)^T \quad -2r_{ij}] \begin{bmatrix} \mathbf{x} \\ d_j \end{bmatrix} = r_{ij}^2 - \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2$$

where the row vector and RHS are known, and $\mathbf{x}$ and d_j are unknown. If we now write the equations for all pairs (i, j), $i < j$ and assume $N = 4$ sensors, we obtain

$$\begin{bmatrix} 2(\mathbf{x}_2 - \mathbf{x}_1)^T & -2r_{12} & & \\ 2(\mathbf{x}_3 - \mathbf{x}_1)^T & & -2r_{13} & \\ 2(\mathbf{x}_4 - \mathbf{x}_1)^T & & & -2r_{14} \\ 2(\mathbf{x}_3 - \mathbf{x}_2)^T & & -2r_{23} & \\ 2(\mathbf{x}_4 - \mathbf{x}_2)^T & & & -2r_{24} \\ 2(\mathbf{x}_4 - \mathbf{x}_3)^T & & & -2r_{34} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} r_{12}^2 - \|\mathbf{x}_1\|^2 + \|\mathbf{x}_2\|^2 \\ r_{13}^2 - \|\mathbf{x}_1\|^2 + \|\mathbf{x}_3\|^2 \\ r_{14}^2 - \|\mathbf{x}_1\|^2 + \|\mathbf{x}_4\|^2 \\ r_{23}^2 - \|\mathbf{x}_2\|^2 + \|\mathbf{x}_3\|^2 \\ r_{24}^2 - \|\mathbf{x}_2\|^2 + \|\mathbf{x}_4\|^2 \\ r_{34}^2 - \|\mathbf{x}_3\|^2 + \|\mathbf{x}_4\|^2 \end{bmatrix}$$

This is an overdetermined linear set of equations (6 equations, 5 unknowns) of the form $\mathbf{Ay} = \mathbf{b}$, and can be solved by computing the pseudo-inverse (or left-inverse) of $\mathbf{A}$, i.e., $\mathbf{y} = \mathbf{A}^\dagger \mathbf{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$.

We thus obtain $\mathbf{x}$, as well as the nuisance parameters d_2, d_3, d_4 that depend on $\mathbf{x}$.

Although there are 6 equations, they are linearly dependent: verify this (e.g., on a noiseless testcase in Matlab).

C.2 MULTI-DIMENSIONAL SCALING (MDS) ALGORITHM

The Multi-Dimensional Scaling (MDS) algorithm is a well-known algorithm for solving the following problem. Given distance information like the following:

	Amsterdam	Arnhem	Den Haag	Groningen	Haarlem	Rotterdam	Utrecht	Zwolle
Amsterdam	0	112	60	178	19	85	47	110
Arnhem	112	0	121	168	119	113	69	66
Den Haag	60	121	0	232	63	33	62	153
Groningen	178	168	232	0	199	248	193	107
Haarlem	19	119	63	199	0	87	54	131
Rotterdam	85	113	33	248	87	0	55	146
Utrecht	47	69	62	193	54	55	0	91
Zwolle	110	66	153	107	131	146	91	0

Plot the positions on the map:

Or, expressed in mathematics: Given N nodes and all distances d_{ij}, compute all locations $\mathbf{x}_i^T = [x_i,\ y_i]$.

The MDS algorithm is derived as follows. Note that

$$d_{ij}^2 = \|\mathbf{x}_i - \mathbf{x}_j\|^2 \ = \ \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j - 2\mathbf{x}_i^T\mathbf{x}_j \tag{C.2}$$

We can collect all distance pairs (squared) into a matrix $\mathbf{D}$, and make the following definitions:

$$\mathbf{D} = \begin{bmatrix} d_{11}^2 & \cdots & d_{1N}^2 \\ \vdots & \ddots & \vdots \\ d_{N1}^2 & \cdots & d_{NN}^2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N^T\mathbf{x}_N \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \quad \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Then the data model (C.2), written in matrix terms, is

$$\mathbf{D} = \mathbf{b}\mathbf{1}^T + \mathbf{1}\mathbf{b}^T - 2\mathbf{X}\mathbf{X}^T. \tag{C.3}$$

Here, $\mathbf{D}$ is known, $\mathbf{X}$ is unknown and to be estimated, and $\mathbf{b}$ is an unknown vector with the distance of each node to the origin, which could be derived from $\mathbf{X}$: it is redundant. This is a set of quadratic equations.

To solve (C.3) using linear algebra, we first try to remove $\mathbf{b}$ from the equations. For this, define a projection

$$\mathbf{P} = \mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{N}$$

Verify that this projection satisfies $\mathbf{P}\mathbf{1} = \mathbf{0}$: it can be used to project the vector $\mathbf{1}$ away. The effect of $\mathbf{P}$ on a vector $\mathbf{v}$ is to remove the "mean" of a vector. Indeed, if we write $\mathbf{v} = \mu\mathbf{1} + \mathbf{v}_0$, where $\mathbf{v}_0$ is a "zero-mean" vector where $\mathbf{1}^T\mathbf{v}_0 = \sum_i (\mathbf{v}_0)_i = 0$, then $\mathbf{P}\mathbf{v}_0 = \mathbf{v}_0$ and $\mathbf{P}\mathbf{v} = \mathbf{v}_0$.

Apply $\mathbf{P}$ to the left and the right of (C.3):

$$\mathbf{PDP}^T = -2(\mathbf{PX})(\mathbf{X}^T\mathbf{P}^T) \tag{C.4}$$

Note that this decomposition implies that $\mathbf{PDP}^T$ is rank 2 (the number of columns of $\mathbf{PX}$).

The next step is to factor $\mathbf{PDP}^T$ using the eigenvalue decomposition. Write

$$\mathbf{PDP}^T = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

Since it is rank-2, we can introduce an "economy-size" decomposition

$$\mathbf{PDP}^T = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^T$$

where $\hat{\mathbf{\Lambda}}$ contains the two non-zero eigenvalues, and $\hat{\mathbf{U}}$ the corresponding two eigenvectors. This decomposition can be written as

$$\mathbf{PDP}^T = (\hat{\mathbf{U}}\hat{\mathbf{\Lambda}}^{1/2})(\hat{\mathbf{\Lambda}}^{1/2}\hat{\mathbf{U}}^T)$$

and if we compare to (C.4), we find

$$\mathbf{PX} = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}^{1/2}\mathbf{Q}$$

for an unknown 2×2 unitary matrix $\mathbf{Q}$. Recall that our objective was to find $\mathbf{X}$, the matrix with locations of all nodes. Instead we find $\mathbf{X}' := \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}^{1/2}$. The above equation shows that $\mathbf{X}'$ is equal to $\mathbf{PX}$, up to an unknown rotation/reflection $\mathbf{Q}$; we cannot invert $\mathbf{P}$ because a projection is singular. The effect of $\mathbf{P}$ is a translation of $\mathbf{X}$ towards the origin, such that the mean (center of gravity) of each column of $\mathbf{X}'$ is zero.

Thus, we can find the (x_i, y_i) locations of the nodes up to a translation and rotation/reflection. This is reasonable, because we start with only distance information. In practice, we would define the location of one of the nodes to be the $(0,0)$ location, and define another node to be in the direction of the positive x-axis. After this, the only unknown is a possible reflection.

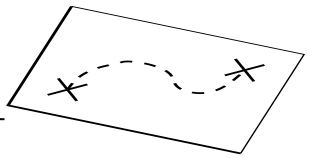
This above constitutes the MDS algorithm. It is frequently used e.g., in psychometrics and sociology. For example, it is sometimes hard to grade students, but we can compare them in pairs to see who is better, and assign some score (= distance) to this. If we have scores on all pairs, we can put them in a matrix $\mathbf{D}$ and apply the algorithm to find grades.

We presented it for the 2-dimensional case, but it can easily be extended to the 3-dimensional (or even N-dimensional) case. For the N-dimensional case, the rank of $\mathbf{PDP}^T$ is N. Thus, this projected distance matrix gives us information on the number of dimensions of the problem. For the student grading problem, we would hope on a rank 1, but if the ordering is not conclusive (some students are better in math, others in english), we may see from the rank that we need higher dimensions. In that case, we find student grades on two (or more) orthogonal aspects.

The algorithm is implemented in Matlab as `mdscale`, but you can easily make your own function so that you exactly know what is going on.

Appendix ***D***

SOFTWARE DEVELOPMENT USING SCRUM



Contents

Scrum and Agile are techniques originally developed for ICT project management. Rather than a very precise product specification and subsequent lengthy development cycle where the end result might not be what the client wanted, the idea is to have a more flexible design cycle where the product is iteratively refined. At the same time, team members are more involved and carry more responsibility.

This appendix describes the scrum development method and its possible integration in the EPO4 project. This tutorial is not mandatory, but can provide a view of how professional teams use a development method to reach a common goal.

D.1 INTRODUCTION TO SCRUM

In the ICT world it is difficult to specify well. Customers and clients don't always know what they specifically want, which can result in changing specifications during a period of time. The scrum development method gives the flexibility to even in late stage adjust to changing specifications. Although the EPO4 project has pre-set specifications, certain limitations of sensors or transmitters discovered during the project can result in a change of strategy or implementation. This will result in the need for new specifications.

The scrum method is a feedback-driven empirical approach. In scrum, the work is developed in short (1-4 weeks) iterative cycles called "sprints". Each time a sprint is completed, a working subsystem is created. If all sprints from the "release backlog" are completed, the product is obtained.

D.2 TASK ASSIGNMENT

1. *Product owner*

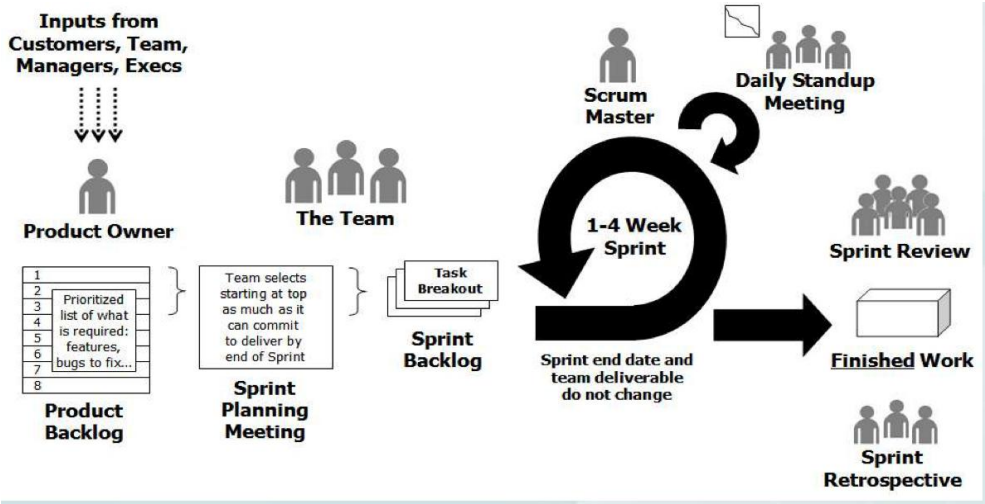


Figure D.1: Overview scrum development method[1]

The product owner is the client or customer. The client or customer has the biggest interest in the product. At the start of the scrum the customer or client writes down "user stories" (wishes from a user perspective) to show what they want from the finished product. The product owner controls the "product backlog" (wish list), he/she determines in what order the user stories are implemented and sorts them on priority. In the context of EPO4, the product backlog is the list of requirements for both the mid-term challenge and the final challenge.

2. *Design team*

   The design team is responsible for delivering the software product at the end of each sprint. The team is small (max 7 persons), multidisciplinary, and self-organised. They fix the analysis, design, development, testing and documentation and make sure that at the end of the sprint a working product can be presented, and when necessary can be taken into production.

3. *Scrum master*

   The scrum master coordinates the team and makes sure that the scrum process is being followed. The scrum master coordinates all meetings with its team members. The scrum master makes sure that the team is not bothered by third party people with extra demands during the sprints, and when someone from the team is needed elsewhere he/she prevents this from happening. The scrum master is not a (hierarchical) project manager. The scrum master doesn't make personal decisions, because these could decrease collaboration and openness between him and the team.

[1] https://www.linkedin.com/pulse/20141021025927-21583419-mobile-development-using-agile-scrum. Last checked on 16 April 2019.

D.3 PROCESS

The scrum development method/framework is aimed to achieve a product within a short period of time. A visualization of the scrum development method is given in Figure D.1.

In scrum, you work with a product backlog. This product backlog describes what user stories should be added to be able to create the product. User stories and are written from user perspectives. The draft of the user story goes as follows:

> As a (role), I want (feature), so that (benefit)

After the gathering of user stories with all people involved (customers, the team, etc.) it is required to prioritize. The product owner selects the user stories which will be used to make the product.

After the user stories are prioritized, the product backlog is broken into one or more release backlogs (e.g., for EPO4 this could be the mid-term/final challenge). These release backlogs are products which can be demonstrated. All the priorities of the product backlog are discussed during the sprint planning meeting and written in more technical terms. This to make it easier for the team to understand what is needed to be done. Due to the complexity of release backlogs, they are further broken down into a number of sprint backlogs: short duration milestones for creating subsystems. In the context of EPO4, a sprint could be a 1-week period.

The progress of each sprint is monitored using the sprint backlog and sprint burn-down chart. A daily scrum meeting ensures everything is on track. After each sprint a retrospective meeting is held to fine-tune upcoming sprints. After the sprint a sprint review is held to show to the product owner what has been accomplished during the sprint.

D.3.1 Before the sprint

It is recommended to put a lot of time on preparation before the sprint, because providing well thought and written user stories in the beginning of the project usually prevents changes later on. Writing a good description of the needed system in the sprint planning is necessary to give a steady release backlog which then can be easier broken into subsystems. The different steps before the sprint are as follows:

1. *Product backlog* The product owner builds a list of features and bright ideas which could be used in the product. The product-owner is the owner of this list and terminates the order. Therefore it is important to fully understand what the product owner exactly wants. Every team member can however add things to the list, but the product owner is and stays responsible. The product backlog also gives an estimation of the time needed to complete the task. The product owner prioritises the list and brings the top items to the team sprint planning team. The product owner and scrum master discuss the top user stories what can go into the release backlog.

[2]Edited from: http://www.agile-scrum.be/wordpress/wp-content/uploads/2014/07/Product-Backlog-Agile-Scrum-Belgium-Training.png. Last checked on 16 April 2019.

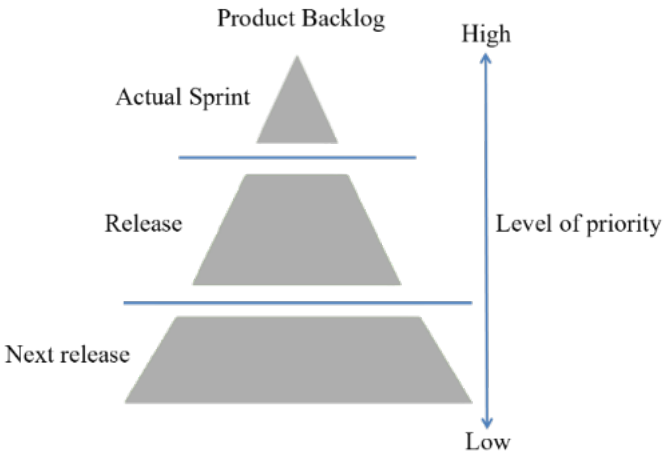


Figure D.2: Product backlog divides user stories in 3 categories [2]

2. *Sprint planning (release backlog)*

   The most important user stories from the product backlog have been combined into a release backlog (Figure D.2). From this release backlog a working product must build. Normally the top, most important stories of the product backlog will be handled in the sprint.

   It is crucial that the design team picks the user-stories, because the team members are the people which do the underlying task and know best what is needed. That is why the team decides how much work can be included in the sprint and is the teams task to estimate the amount of work per user-story. The design team must confirm that everything is clear. After confirmation the tasks get broken down and divided in the sprint backlog.

3. *Sprint Backlog*

   The sprint backlog defines what is needed to complete the sprint. The sprint backlog is composed of the top items of the product backlog and summarized in a release backlog. The design team determines how to divide the task, keeping in mind all members strong and weak skills. Most of the time the tasks are given to the people which did these task the fastest/best based on the last sprint. This increases team involvement, enthusiasm, motivation and making the problem their own. Usually the items on the sprint backlog are written post-its. There are four columns:

   To do || Doing || Testing || Done.

   This gives every team member a clear overview what has already be done and how far in the project they are. After the start of the sprint nobody from the outside can add things to the board. After the complement of the sprint the product backlog is be revisited and priorities can be adjust and changes can be made.

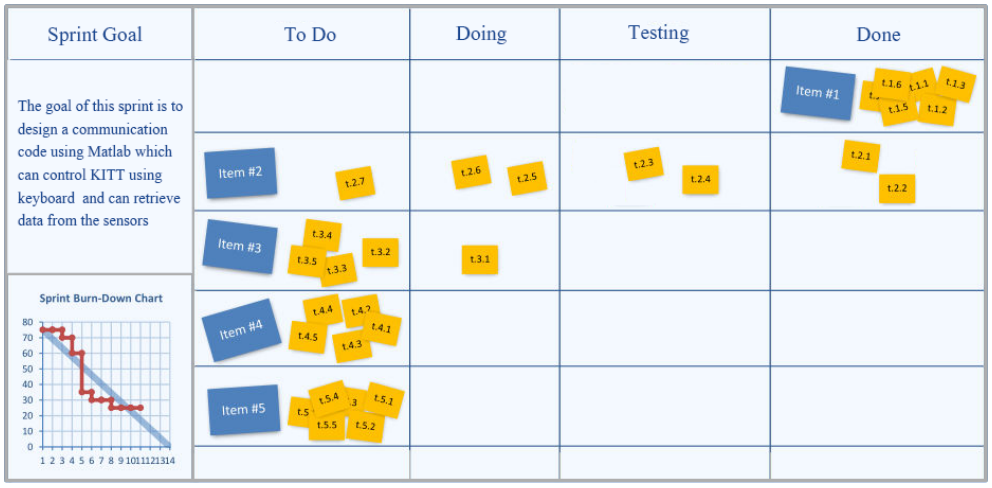


Figure D.3: Sprint planning and burn-down chart[3]

D.3.2 During the sprint

The main task during the sprint is to build the subsystem that has been discussed and documented in the sprint planning. There are however certain regulations that are needed to be done besides the building. These are written down below:

1. *Burn down chart*

   The burn down chart of the sprint planning (figure D.3) shows the team how much works needs to be done in order to complete the sprint. This gives a clear overview if team needs to hurry up or more time can be spend at the finishing of the sprint.

2. *Daily stand-up meeting* Every project day a meeting must be held. Everybody in the meeting needs to stand. This improve quickness and decrease endless discussions. In the meeting the following three questions are asked to each member of the group:

   - What have you done;
   - What are you going to do today;
   - Are you facing any difficulties or challenges, do you need help with these challenges, and are these challenges of importance for the rest of the team.

   This meeting can't take more than 15 mins. All of the bigger problems are discussed outside of the meeting. This to let those who aren't involved being able to get back to work.

[3]Edited from: http://mayakron.altervista.org/wikibase/show.php?id=Scrum.

D.3.3 After the sprint

When the sprint is completed a working subsystem is created. To be able to improve the sprint the following actions have to be taken:

1. *Sprint review*

   In the sprint review the working subsystem will be presented. If not working fully or correctly this will be addressed. During the EPO4 project after every sprint the subsystems are written down in the mid-term/final reports.

2. *Evaluation (Retrospective)*

   The evaluation is meant to learn what went well and what went wrong as a clear goal for the team how to improve even more. All team members must attend this evaluation. The meeting will be led by the scrum master. All the team members tell what went wrong and what went correctly. Not only these points are adjusted but also the most surprising or most difficult object can be discussed. It is not meant for personal attacks or blaming. Everybody should learn something from this for the oncoming sprints. This creates a consequent feed of feedback.