EE2L21

Manual

EPO-4: Autonomous driving challenge

2024 Edition

Delft University of Technology

Prof.dr.ir. Alle-Jan van der Veen
Dr. Bahareh Abdikivanani

TU Delft Delft University of Technology

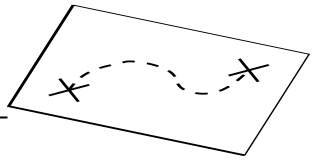
Challenge the future

The Project KITT: Autonomous Driving Challenge (EPO4) was conceived in 2011 as a semester project for second-year BSc-EE students. It has since then seen many revisions. Thank you to all who have contributed to it.

Contents

Chapter 1

INTRODUCTION



Contents

Welcome to the EE2L21 EPO-4 project option "KITT: 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. The car called "KITT" can communicate to a base station (your computer) via Bluetooth and has a pair of distance (anti-collision) sensors and an audio beacon on top. During a final competition, you are asked to demonstrate your achievements in several "challenges", i.e., races of increasing difficulty.

Like in the previous projects, the objective of EPO-4 is to apply and integrate the EE knowledge you have acquired so far. This entails signal transforms, telecommunication, system modeling and control, and digital signal processing.

This manual starts with a description of the overall project. This complex task is split into modules, each contributing to the overall goal. You have already designed a TDOA distance estimation algorithm in the lab sessions of EE2T11 Telecommunication 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 detect obstacles in front of it, and can communicate with a base station (your PC), which performs calculations such as location estimation, trajectory tracking, and collision avoidance (see figure 1.1). You will have to write these functionalities. You will also have to document the designs in

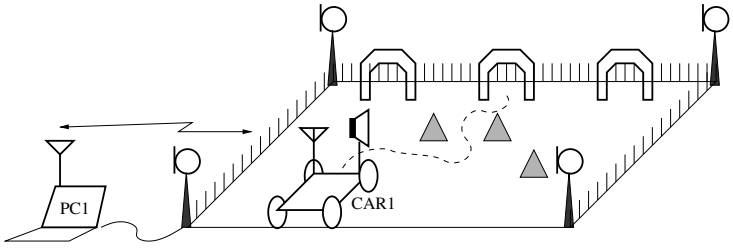


Figure 1.1: General set-up

your reports, describe the design choices, and evaluate the test results. Lastly, you will have to present and defend your reports and designs.

At the start of the final challenge, the car is positioned at a known location. The car should drive autonomously to the target while avoiding obstacles like boxes and competing cars. The car is equipped with anti-collision sensors and a localization beacon. A bidirectional RF link will provide a connection of the car to a base station (PC) that allows the transfer of the anti-collision ranging measurements, and to receive actuation instructions (motor control and control of the equipment on the car).

This is a complex assignment, which is therefore split into several modules:

- Communicating with KITT (Modules 1 and 2)
- Localization using microphones (Module 3)
- Car model (Module 4)
- State tracking and control (Module 5)
- System integration.

An overview of the entire system is shown in Fig. 1.2. In the control loop, the virtual car model ("digital twin") predicts the car position (and its velocity and orientation), and this is used to assist in tracking the state without independent position estimate from the microphones. This block is not absolutely necessary; however, without it, you probably need a pretty good localization system.

The total duration of the project is 9 weeks. Each week, you have two lab sessions; outside of the lab, you must prepare the lab assignments, homework, and report writing. The project is finished with the Final Challenge (demo session, week 9) and the Presentation and Defense (committee interview, week 10).

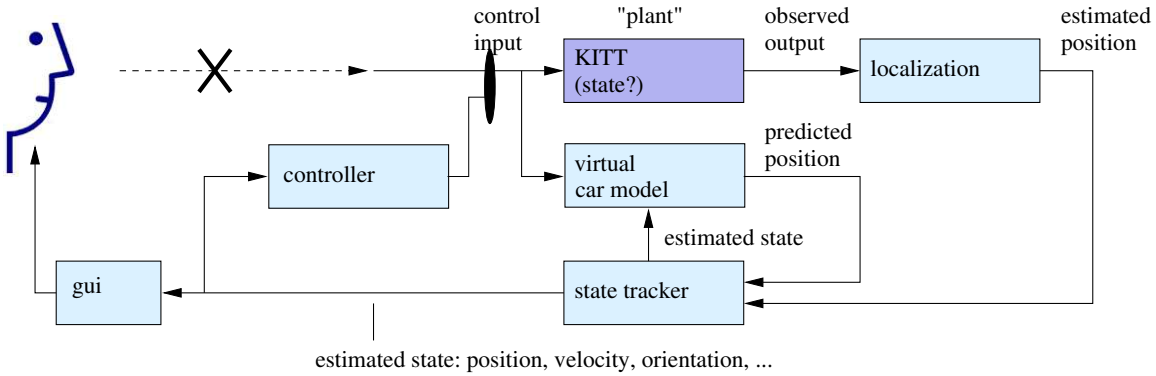


Figure 1.2: System overview

1.2 DESIGN SPECIFICATIONS

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

1. The *Connecting to and controlling KITT* module consists of setting up an RF connection to KITT and writing a program that allows you to control KITT and drive it around. The connection is provided by off-the-shelf Bluetooth modules, controlled by a microcontroller on the car and an interface on the base station PC. The microcontroller can also generate motor control signals using basic commands from the base station PC.

2. The *Reading KITT sensor data and detecting objects* module consists of reading and using the data from the anti-collision sensors and the microphones along the field. These distance sensors are a pair of off-the-shelf ultrasonic transmitters/receivers that can detect objects in front of them.

    - The maximal range of the detection is about 6 m.
    - The system is mounted on the car.
    - The system is controlled by the onboard microcontroller and read out via the RF link.

3. *Localization using audio communication* is done by transmitting a repetitive beacon signal from the car. This is an audio signal, and it may be corrupted by interfering signals emitted by other cars present in the field (depending on whether you choose to do this challenge), and by surrounding noise.

    - The audio signals from all cars are captured by multiple microphones placed along the field. The analog microphone signals are sampled and made available to you.
    - There are 5 microphones around the field, of which the location is known.
    - The estimation of the location is done in Python by using the microphone data and the TDOA you developed in the EE2T11 Telecommunications A practicum.

- The location updates should be frequent and accurate enough for the trajectory tracking to function. (Minimal numbers are at least one update per 2 seconds, with an accuracy better than 30cm.)

4. *Control and trajectory tracking* is based on a sufficiently accurate system model of the car, both regarding velocity and steering behavior.

   - The state of KITT consists of position, velocity, and orientation (4 parameters).
   - The velocity model is based on differential equations that take the speed settings into account.
   - The steering behavior is based on modeling the trajectory resulting from a steering command.
   - The model of the car should allow you to predict the next state of the car, given the current state and the control input.

5. *System integration for the final challenge* brings your previously done work together. KITT's starting location and orientation are known, and from this you will have to route KITT to certain points in the field.

   - The system you make must be closed loop, control input is from your localization algorithm that uses an audio beacon on top of the car, additionally you may have to use the data from the anti-collision sensors. It is important to note that "open loop" solutions will not be accepted. These types of solutions have hard-coded timings or control actions that can only address a single scenario.
   - The control actions are computed in Python on your base station PC.
   - After the start of the challenge, you are not allowed to touch the base station PC.
   - The main considerations in the design are the proper managing of uncertainties in localization, state updates, and real-time aspects: the car drives while estimating its location.

Each of these functionalities is specified in its own module.

1.3 TEAMWORK DIVISION

Effective teamwork is crucial for the success of the project. Given that your team consists of four or sometimes five students, it is advisable to divide the work into parallel work packages. For the first weeks, we suggest to follow the Modules, e.g. *Communicating with KITT* and *Localization algorithm development*. In the second phase of the project, a path planning algorithm has to be created, which takes the location of the car and the target as input and generates a set of waypoints to get there. The control part should then send commands to KITT to drive to these locations. A detailed car model will help to develop and debug your code.

1.3.1 Communicating with KITT

Two team members can focus on the communication aspect. This involves establishing a reliable and efficient communication link between the computer (base station) and KITT. Tasks include:

- Implementing communication over Bluetooth.
- Developing functions to send commands to KITT.
- Creating modules to read sensor data from KITT.
- Characterizing the sensors and motion of KITT.
- Make recordings using the microphones.

Communication with KITT is discussed in Modules 1 and 2.

1.3.2 Localization algorithm development

The other two team members can focus on the localization algorithm. This involves processing audio data from the microphones to estimate the location of KITT. Tasks include:

- Developing a channel estimation algorithm for received audio signals. (A basis for this has been designed during the course labs of Telecommunications A)
- Implementing Time Difference of Arrival (TDOA) calculations.
- Using TDOA data locate KITT
- Handling data synchronization and ensuring accuracy in localization.

This is further explained in Module 3.

1.3.3 Collaboration

You maximize efficiency and expertise by assigning dedicated team members to each area in parallel, as only so many people can type on a keyboard simultaneously. Remember that effective communication within the team is critical. Regular updates, meetings, and shared documentation will help keep everyone on the same page and facilitate successfully integrating the individual components into the autonomous driving system. To better understand what the other group is working on, it is advisable to read the manual on the other modules.

A possible approach to organizing teamwork follows the “Scrum” or “Agile” methodology. In this approach, a project evolves iteratively, producing a functional system in each iteration (“sprint”). A description of the Scrum method is given in Appendix D.

You will find that working in parallel is only possible if you define clear interfaces: e.g., route planning is only possible if it is clear how accurate the localization algorithm works. In turn, the localization algorithm must know time stamps to know how "old" the microphone data is, and produce a time stamp telling how old the location fix is.

1.4 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 consist of advanced Python programming.
- Increased skills in measurement techniques, e.g., wireless channel measurements.
- Application of course material from various courses: control systems, linear algebra, signal transformations, digital signal processing, telecommunication.
- 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.

The main difficulties students have with this project are the complexity of the overall system and planning. You will split up your team into two sub-groups. Each delivers Python code that later has to be integrated and function together. Nobody on the team has complete, detailed knowledge of the entire design. If you don't manage this, then you will fail. Essential parts to make this work are:

- *Sytem Engineering:* At an early stage, agree on an overall structure for your design, partition it into modules, and decide on specifications for the modules. Find the essential aspects that need to be done right. For this system, uncertainty in time and position plays an important role. E.g., the car moves while the localization is being computed.
- *Testing:* Each module should have a clearly defined specification, and the Python code should be tested and verified against the specifications before it is integrated into the overall system. You cannot debug the overall system if you are unsure about its parts!
- *Planning:* You will probably find that there is insufficient time in the testing areas: it is available to your team only about 25% of the time, and with two sub-groups, you will also compete within your team. Thus, you must plan well on what you want to measure and test during your time slot in the testing area.
- *Modeling:* To avoid waiting for precious testing time, it is advisable to create a (simple or more advanced) software *car model*, such that you are able to test your algorithms on the model. This also allows you to verify in detail the performance of your algorithms.

Table 1.1: Generic schedule and deadlines

Date	Description
Week 1	Kick-off + getting your IDE setup + Start on modules 1, 2, 3
Week 2	Work on modules 1, 2, 3 + Start on module 4
Week 3	Complete modules 1, 2, 3
Week 4	Complete module 4 + Midterm report writing + Sign-off modules 1, 2, 3
Week 5	Sign-off module 4 + **Midterm report deadline**
Week 6	System integration, prepare for challenge A
Week 7	Testing challenge A
Week 8	Complete challenge A, start on challenges B and C; final report writing
Week 9	**Final challenge + Final report deadline**
Week 10	**Presentation and discussion**

1.5 SCHEDULE AND DEADLINES

EPO-4 has 14 scheduled lab sessions (distributed over 9 weeks due to holidays). Outside of these sessions, you will have time to prepare and work on the report. In week 9, there is a common session for the final challenge, and in week 10, you will be called for a final presentation followed by a discussion. The total lab time is 56 hours, and preparation/homework time is budgeted at 84 hours, for a total study load of 5 EC.

Table 1.1 shows a generic schedule. For the specific dates this year, look in Brightspace! The planning regarding the completion of the modules is flexible, but the deadlines are firm.

Many groups compete for time on the test fields: 4 to 5 groups share the same field. You must reserve time slots on one of these systems using a planner in Brightspace. Come prepared with a measurement/test plan!

1.6 ASSESSMENT AND REPORTS

1.6.1 Mid-term assessment and report

In weeks 4 and/or 5, you are asked to demonstrate the functionality and performance of your communication, localization and car model scripts to your TA. After their approval, you are ready to write your mid-term report, detailing your approach, implementation, and test results for Modules 1–4.

Sign-off Modules 1,2: Communication script assessment Using your communication script, you are asked to demonstrate that you can control your car using the WASD or arrow keys on the keyboard. You should be able to transmit the beacon signal and record it using the 5 microphones. The detailed requirements are outlined in Modules 1 and 2.

Sign-off Module 3: Localization script assessment Using your localization script, you are asked to demonstrate that you can locate the car from recordings that are given to you. (Integration with your own car is not yet necessary.) Refer to the specifications outlined in Module 3 for detailed requirements.

Sign-off Module 4: Car model assessment In Module 4, you are asked to write a basic car model that captures the dynamics of KITT: its response to driving and steering commands. You do this based on measurements of its behavior, along with simple equations of motion. The goal of this model is to be able to predict the location of KITT after a few seconds of driving. You can also use this model to test your control algorithms.

You demonstrate your car model by predicting the position of KITT after a short series of commands, and by comparing this to the actual outcome.

Mid-term report You submit a report of approximately 15 pages (plus cover page and appendix) documenting your work until now. More details on the structure and contents of this report are in Chapter 7.

1.6.2 Final challenge and report

Final challenge To conclude the project, your design is tested during the final challenge. The tests will be described briefly here and in more detail in Chapter 9.

The final challenge starts at a known location and orientation. At a start signal, you have to drive autonomously to a target. If you complete the task successfully, you may take on more complex tasks, including additional waypoints, obstacles, and other cars. It is required to reach the target with a certain minimal accuracy. Bonus points are obtained by the fastest team for each challenge. During the race, it is not allowed to touch the car.

Speed is not the main requirement in this challenge. You may reach the target faster by driving faster, but navigating becomes more complex, location updates must be computed more quickly, and you have less time to avoid obstacles.

Final report The project outcome is documented in a final report. The report must follow a structured approach, which you have learned in previous projects; the midterm report is a part of it. Use a to-the-point, concise, but complete reporting style. Provide an appendix with all Python code. More details on this report are in Chapter 9. Your report is submitted in the corresponding submission folder in Brightspace.

The report without an appendix should be about 30 pages. Within these pages, you must document your design choices, explain your control systems, and report the deliverables of all modules, how you combined these modules, and what problems you ran into and solved them. The focus is on your findings and measurement results and the corresponding conclusions. You are also judged regarding project skills such as planning and teamwork.

Final presentation and discussion In week 10 (consult Brightspace for the exact date), you present and defend your final report before an examination committee. The examiners will ask questions about your design choices and aspects of teamwork. This will be part of your grade.

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.

While each team member may not have been directly involved in every aspect of the project, through open communication and collaboration, all team members are expected to have an operating understanding of all parts of the project. During the discussion, detailed questions will be asked to the group. The team member with the best subject knowledge is encouraged to answer. However, less complex questions can be asked to anyone to test their participation in the project.

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.

1.6.3 Grading

Your grade depends on the following:

- Mid-term report (30%);
- Performance during the final challenge (20%);
- Final report (35%);
- Oral presentation and defense (15%).

Teamwork is important. Your individual grade may differ depending on staff observations and peer reviews. There is a penalty for submitting the report late.

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

1.7 FACILITIES

The project is carried out at the Tellegen Hall facilities in A and B teams. Each team has 2 sessions per week. The hall has three testing areas, each shared by 4 teams.

The following support is available:

- *Student assistants*; student assistants are your primary help. Each assistant supports up to four teams. Assistants also check attendance and progress.

- *Instructors*; Practicum coordinators are available at each lab session. The coordinators also grade your reports.
- *Technical support*; for questions about hardware and implementation issues, you can contact the student assistants and/or the technicians at the facilities.

Visit the Brightspace page of the course for support and contact information.

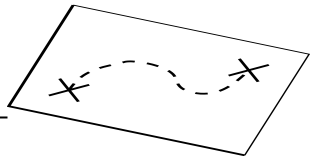
1.8 RULES AND REGULATIONS

In addition to the rules and regulations of the EEE group on the use of the Tellegen Hall, the following rules and regulations are applicable:

- You are expected to be present during your scheduled lab sessions. If you cannot attend for a good reason, contact your assigned TA *before* the lab session. Absence more than two times will not be allowed; you will be removed from the practicum.
- Preparation for labdays is *mandatory*. This project has an intense pace and limited time; use it wisely. The scheduled homework time is needed.
- You may not work alone in the lab. A student assistant or staff member must be present.

Chapter **2**

BEFORE WE START



Contents

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 assignment 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.

One thing to decide is to form two sub-groups: one that will work on localization (Module 3) and the other to work on the car interface (Modules 1 and 2). After these modules are finished, Module 4 (car model) must be completed, and the mid-term report must be written according to the guidelines in Chapter 7.

Before we can start doing anything, you need to set up your programming environment in Python. If you have not done so before, this chapter will guide you on the essential steps to installing Python on your laptop or PC. We also need an Integrated Development Environment (IDE), and some required packages for the project.

We will use Object-Oriented Programming (OOP), and the keyboard and timer package, a short tutorial is provided here as well.

2.2 INSTALLING PYTHON

The first step in setting up your Python environment is to install the Python interpreter. The installation process is slightly different for Windows, Linux, and Mac. For the best compatibility, everyone on the team should install the same version of Python. As of 2024, it is recommended that Python 3.12 be installed.

2.2.1 Windows

For Windows users, follow these steps to install Python:

1. Open the Microsoft store, and search for "Python". Install the latest version.

2. To verify the installation, open a command prompt and type:

   ```
   python --version
   ```

   This should display the installed Python version. Please verify that the installed version matches the one you downloaded from the Microsoft store. If not, follow this tutorial: How to Add Python to PATH, or ask a TA for help.

2.2.2 macOS

For macOS users, an old verion of Python is pre-installed, do not use this. Python3 needs to be installed, for this follow these steps:

1. Open your web browser and navigate to the official Python website at https://www.python.org/downloads/.

2. Click on the ”Downloads” tab, and you will see a button for the latest version of Python. Click on it to initiate the download.

3. Once the installer is downloaded, run the executable file. The installer will walk you through a wizard to complete the installation, and in most cases, the default settings work well, so install it like the other applications on macOS. You may also have to enter your Mac password to let it know that you agree with installing Python.

4. To verify the installation, open a command prompt and type:

   ```
   python3 --version
   ```

2.2.3 Linux

For Linux users, an old version of Python is often pre-installed, do not use this. You can use the package manager specific to your distribution to install Python. Follow these steps:

1. Open a terminal.
2. Update your package manager's repository information:

   ```
   sudo apt update  # For Debian/Ubuntu
   ```

3. Install Python:

   ```
   sudo apt install python3  # For Debian/Ubuntu
   ```

4. To verify the installation, type:

   ```
   python3 --version
   ```

   This should display the installed Python version.

2.3 INSTALLING AN IDE

It is recommended that you use Visual Studio Code (VSCode) as your Integrated Development Environment (IDE). VSCode is a free, open-source code editor developed by Microsoft with a wide range of extensions and excellent Python support.

Please note that you are allowed to use another IDE, such as PyCharm (you can get a premium account using your @student.tudelft.nl email address).

2.3.1 Windows, macOS, and Linux

1. Open your web browser and navigate to the official VSCode website at https://code.visualstudio.com/.
2. Click on the "Download" button to download the installer suitable for your operating system.
3. Once the installer is downloaded, run it to start the installation process.
4. Follow the on-screen instructions to complete the installation. You can choose the default settings unless you have specific preferences.
5. After the installation is complete, open Visual Studio Code.

2.3.2 Installing Python extension for VSCode

VSCode is very popular because of its use of extensions. This makes it compatible with almost any programming language or development board (such as Arduino). In our case, you have to add Python support to the IDE. Follow these steps:

1. Open VSCode.
2. Go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window.
3. In the Extensions view search bar, type ”Python”.
4. Look for the official Python extension by Microsoft and click ”Install”. This extension provides enhanced support for Python development.
5. After installation, you may need to reload VSCode to activate the Python extension fully.

2.4 INSTALLING REQUIRED PACKAGES

Python is a well supported language with many possibilities. These possibilities come from the many libraries and packages that are built by users. Because we do not want to rewrite all the essentials, we will use a list of required packages that every user has to install.

To install these packages, go to the directory were you want to work on EPO-4. In this directory paste the "requirements.txt" file from Brightspace. Follow the instructions based on your system:

2.4.1 For Windows 11:

Open a terminal in your EPO-4 directory and run the following command:

```
pip install -r requirements_student.txt
```

2.4.2 For Windows 10:

Open a terminal in your EPO-4 directory and run the following command:

```
pip install -r requirements_student.txt
```

The appropriate sound card driver must be used to use the microphone array. The soundcard used in EPO-4 is a PreSonus AudioBox 1818VSL. On Windows 10, it is necessary to install ASIO4ALL (https://www.asio4all.org/) and a build of PyAudio compiled with ASIO support (https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyaudio).

2.4.3 For MacOS:

Open a terminal and install Homebrew using:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)
```

It will ask you for confirmation, and your password. If prompted run the follow up commands. Afterwards, install portAudio using Homebrew, this fills the gaps in some packages that are needed to use the microphones:

```
brew install portaudio
```

Open a terminal in your EPO-4 directory and run the following command:

```
pip3 install -r requirements_student.txt
```

2.4.4 For Linux:

Open a terminal in your EPO-4 directory and run the following command:

```
pip3 install -r requirements_student.txt
```

2.5 OBJECT-ORIENTED PROGRAMMING IN EPO-4

Throughout the project, you will write over 20 functions and a couple hundred lines of code. As you will find, there is exponential difficulty with the increase in project size. Writing many functions is part of the assignment, but the challenge is testing these functions and getting them to work together. You must communicate with the car and interpret the data received, accurately find the car's location, plan how to drive to the final destination, generate steering commands, and adjust the plan while you drive and discover objects.

This tutorial introduces object-oriented programming (OOP) concepts in Python, which is a programming method that provides flexibility. It is highly recommended that you learn how to use OOP, which will be useful throughout this project and future projects. You can find the code used in this tutorial on Brightspace. The manual will provide code examples, assuming you understand basic OOP concepts to enhance the functionality of the KITT car project.

2.5.1 Class and object

In OOP, a class is a template or a set of instructions for creating objects. On the other hand, an object is a specific instance or realization of that template. To illustrate, let's make a blueprint for KITT using a class.

```
class KITT:
    def __init__(self, model, color):
        self.model = model
```

```
        self.color = color
        self.is_engine_on = False

    def start_engine(self):
        self.is_engine_on = True
        print(f"{self.model}'s engine started.")

    def stop_engine(self):
        self.is_engine_on = False
        print(f"{self.model}'s engine stopped.")
```

Attributes: These are characteristics or properties that describe the state of an object. In the real world, consider attributes as the features defining an object. In the class definition, attributes are represented by variables. In the example, we have defined a class 'KITT' with attributes 'model', 'color', and 'is_engine_on'. Which are just some characteristic of KITT we could define.

Methods: These are functions that define the behavior of an object. They represent the actions that an object can perform. Methods are defined within the class and are used to manipulate the object's state (attributes) or perform some action associated with the object. Continuing with the car example, the methods 'start_engine' and 'stop_engine' control the car's engine.

Self: Within the class definition, 'self' is used to refer to the object.

The __init__method is a special method called the constructor. It is automatically called when a new object is created. In this method, we initialize the `model`, `color`, and `is_engine_on` variables to the values passed.

We can now make some instances of the class KITT. We call these objects.

```
if __name__ == "__main__":
    car1 = KITT("TRX4", "black")    # Make the first instance of KITT
    car2 = KITT("Rustler", "red")   # Make the second instance of KITT

    car2.color = "blue"             # Change the color of car2

    car1.start_engine()              # Start the engine of car1
    print(car1.is_engine_on)         # Output: "True"
    print(car1.model)                # Output: "TRX4"
    print(car1.color)                # Output: "black"
```

First, two instances of KITT are made. They are made from the same KITT class (template) but are a different model and color. These are stored as attributes to the instance (also called object). It is possible to change an attribute of an object after it has been made. In this example, the color of the second car is

changed to blue. The state of the engine is also stored as an attribute. It is defaulted to False (the engine is off). Now, the engine of car1 is started. When checked, car1 outputs that the engine is now set to on.

2.5.2 Method vs Function

In Python, both methods and functions are blocks of code that can be called upon to perform specific tasks. However, there are fundamental differences between the two.

Function:

A function is a block of code that is defined outside of a class and can be called independently. It takes input arguments (if any), performs some operations, and optionally returns a result. Functions in Python are versatile and can be reused across different parts of a program.

Method:

A method, on the other hand, is a function that is associated with an object. It is defined within a class and operates on the data associated with the class (attributes). Methods are accessed through instances of the class (objects) and can modify the state of the object. The first argument of a method is always the special variable 'self', which refers to the instance of the class.

```
class KITT:
    def __init__(self, model, color):
        self.model = model
        self.color = color
        self.is_engine_on = False

    def start_engine(self):
        self.is_engine_on = True
        print(f"{self.model}'s engine started.")

    def stop_engine(self):
        self.is_engine_on = False
        print(f"{self.model}'s engine stopped.")

def drive():
    print("KITT is driving.")

if __name__ == "__main__":
    car = KITT("TRX4", "black")    # Create an instance of KITT
    car.start_engine()             # Call the start_engine method
    drive()                        # Call the drive function
```

In this example, 'start_engine' and 'stop_engine' are methods because they are defined within the KITT class and operate on the KITT object's state. On the other hand, 'drive' is a function defined outside of the class and can be called independently.

2.5.3 Hidden and Private Variables

In object-oriented programming, there are concepts of encapsulation and data hiding, which allow for better control over the accessibility of attributes and methods within a class. This helps in preventing accidental modification of data and ensures the integrity of the class.

Hidden Variables:

In Python, variables and methods can be hidden from the outside world using a single underscore (_) prefix. Although they can still be accessed, it indicates to other programmers that these elements are intended for internal use and should be treated as such.

```
class KITT:
    def __init__(self, model, color):
        self.model = model
        self.color = color
        self._is_engine_on = False  # Hidden variable

    def start_engine(self):
        self._is_engine_on = True
        print(f"{self.model}'s engine started.")

    def stop_engine(self):
        self._is_engine_on = False
        print(f"{self.model}'s engine stopped.")
```

In this modified version of the KITT class, the variable `_is_engine_on` is prefixed with a single underscore. This indicates that it's a hidden variable. While it can still be accessed from outside the class, the underscore serves as a convention to signal that it's intended for internal use.

Private Variables:

Python also supports the concept of private variables, which are variables that cannot be accessed or modified from outside the class. They are denoted by a double underscore (__) prefix.

```
class KITT:
    def __init__(self, model, color):
        self.model = model
        self.color = color
```

```
        self.__is_locked = True  # Private variable

    def lock_car(self):
        self.__is_locked = True
        print(f"{self.model} is locked.")

    def unlock_car(self):
        self.__is_locked = False
        print(f"{self.model} is unlocked.")
```

In this version, the variable `__is_locked` is prefixed with a double underscore, making it a private variable. Private variables cannot be accessed directly from outside the class. Attempting to do so will result in an AttributeError. Special methods should be made to modify these variables called getters and setters.

2.5.4 Getters and Setters

In object-oriented programming, getters and setters are methods used to access and modify the private or hidden variables of a class, respectively. They provide controlled access to the class's attributes, allowing for validation and encapsulation of data.

Getters:

Getters are methods used to retrieve the values of private or hidden variables. They provide a way to access the state of an object without directly exposing its attributes.

```
class KITT:
    def __init__(self, model, color):
        self.model = model
        self.color = color
        self.__is_locked = True

    # Getter for is_locked
    def is_locked(self):
        return self.__is_locked
```

In this modified KITT class, a getter method `is_locked()` is added to retrieve the value of the private variable `__is_locked`.

Setters:

Setters are methods used to modify the values of private or hidden variables. They provide a way to update the state of an object while enforcing validation rules or constraints. For example:

```
class KITT:
    def __init__(self, model, color):
        self.model = model
        self.color = color
        self.__is_locked = True

    # Getter for is_locked
    def is_locked(self):
        return self.__is_locked

    # Setter for is_locked
    def set_lock_status(self):
    if self.__is_locked:
        self.__is_locked = False
        print(f"{self.model} is unlocked.")
    else:
        self.__is_locked = True
        print(f"{self.model} is locked.")
```

In this updated KITT class, a setter method `set_lock_status()` is added to modify the value of the private variable `__is_locked`. This setter will automatically switch the locked state of the car. If it was unlocked, it locks the car, and vice verso.

Example:

```
if __name__ == "__main__":
    car = KITT("TRX4", "black")   # Create an instance of KITT

    # Using getter to check if the car is locked
    print(car.is_locked())        # Output: True

    # Using setter to unlock the car
    car.set_lock_status()    # Output: "TRX4 is unlocked."
```

In this example, first the getter method `is_locked()` is called to check if the car is locked. Then, the setter method `set_lock_status()` is called to change the lock status, because the car when initiated is locked, it is now unlocked.

2.6 PROGRAMMING IN PYTHON

2.6.1 Hints on programming

The above tutorial briefly introduced OOP in Python and demonstrated its application in the KITT car project. However, some of these concepts can be abstract when first introduced. Therefore, you should look for more resources as you experiment with OOP.

- When writing the code to implement functionalities required for this project, partition the code into separate functions and always include a header block with a function. In this header block, you should briefly describe the function, the required inputs, and what the output will do. Including a changelog with author names and dates is also good practice.
- Use meaningful variable names and write many comments so that others can understand what the code is doing.
- Define variables for constants such as `Fs` rather than using numbers throughout the code. That way, you give meaning to that number; if the number changes, you have to change it only at a single location.
- Avoid the use of globals. If a function needs a parameter, include it in the function call. If you must use globals, write the variable names in capitals. The risk of using globals is that if their value changes, it affects functions that depend on them while that dependency is hidden.
- When writing your code, be sure to decouple each function, test it separately, and briefly report on these tests. If you run into any problems further down the design process, finding where functions might not agree and where your problem could lie will be easier.

  The test itself should also be in a script, so that you can frequently rerun it in case some of the functions have changed and need to be tested again.
- In your report, describe the overall structure of the code and the main variables so that others can quickly find their way into your code.
- Test every function in your code! For every 'if' statement in the code, you have two branches to test.

2.6.2 Useful modules

Time Measurement in Python In EPO-4, accurately measuring time is crucial for various tasks, such as determining the duration of events or operations. Python provides the `time` package, which offers functionality to measure time intervals.

To measure time intervals using the `time` package, follow these steps:

```python
import time

# Record the start time
start = time.time()

# Perform an operation or task
# For example, simulate a computational task
for _ in range(1000000):
    pass

# Record the end time
end = time.time()

# Calculate the duration of the operation
duration = end - start

# Print the duration
print(f"The operation lasted for {duration} seconds.")
```

In this code snippet, the `time` package was imported. The `time.time()` function returns your computers time in seconds. The `time.time()` is called to record the start time before executing the operation. After completing the operation, the end time is recorded. By subtracting the start time from the end time, the duration of the operation is calculated.

Detecting Keyboard Events in Python In module 1, you will be tasked with controlling the car from your keyboard. For this you will need to detect keyboard events, such as key presses.

To detect keyboard events using the `keyboard` library, follow these steps:

```python
import keyboard

# Define a callback function to handle key events
def on_key_event(event):
    if event.event_type == keyboard.KEY_DOWN:
        print(f"Key '{event.name}' pressed.")

# Register the callback function for key events
keyboard.on_press(on_key_event)

# Keep the program running to capture key events
```

```
# Use a loop or a blocking function call, or simply wait
input("Press Enter to exit...")
```

In this code snippet, the `keyboard` library is imported. A callback function `on_key_event` is defined to handle key events, this function will be called automatically if a key press is detected. Inside the callback function, the program checks if the event type is `KEY_DOWN`, indicating that a key is pressed.

For this, you need to register the callback function using `keyboard.on_press()`. This function sets up a listener for key press events and calls the specified callback function whenever a key is pressed.

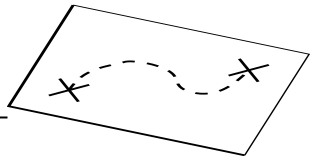
Finally, to keep the program running and capture key events continuously, input prompt is defined to wait for user input. You can exit the program by pressing the Enter key.

Note:

The `keyboard` library provides various functions and event types for handling keyboard events, including key presses, releases, and combinations.

Chapter ***3***

MODULE 1: CONNECTING WITH AND CONTROLLING KITT



Contents

3.1 OVERVIEW

KITT is a remotely controlled vehicle. To allow for this remote control, a wireless link must be used. Over this wireless link, control commands can be sent from the base station (your computer) to KITT. This wireless link also facilitates requests for data from KITT and KITT's data in response to these requests.

This project requires you to maneuver KITT in all sorts of ways. In this module, you will set up a connection to KITT and work with basic controls to allow for more advanced actions later on.

This module starts with an assignment to get to know your team, set up your development environment, and start working with the car.

Learning objectives The following is learned and practiced in this module

- Getting to know your team
- How to set up a Bluetooth connection to KITT
- How to send commands to KITT
- Write a script to drive KITT using keyboard commands

Deliverable

- In the Brightspace assignment folder: submit a filled-out document epo4_kickoff.docx describing how you want to collaborate as a team.
- Basic scripts for driving KITT around using keyboard commands.

Preparation

Figure 3.1: The Traxass E-MAXX and its dimensions

- Read this module
- Look at the KITT commands described here.
- Make sure your Python IDE is ready to go!

Time duration One-and-a-half lab session and one preparation session at home.

What is needed

- KITT, with fresh batteries
- Laptop/PC running Python
- KITT datasheets, found on Brightspace
- `epo4_kickoff.docx`, found on Brightspace

3.2 GETTING TO KNOW KITT

At the start of a project, it's essential to familiarize yourself with the hardware. In this case, that is KITT.

3.2.1 Hardware

KITT is based on a repurposed toy car model, the Traxass E-MAXX (see figure 3.1). 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 components are combined on a single custom-made MCU board installed on KITT (see figure 3.2). The microcontroller forms the core of this board, an NXP LPC4357 chip with an ARM CORTEX^{TM}-M4/M0 core. The MCU board contains the Bluetooth module, connectors for all peripherals, and an amplifier for the audio beacon, which will be used for the localization. Figure 3.3 shows a schematic overview of the connection.

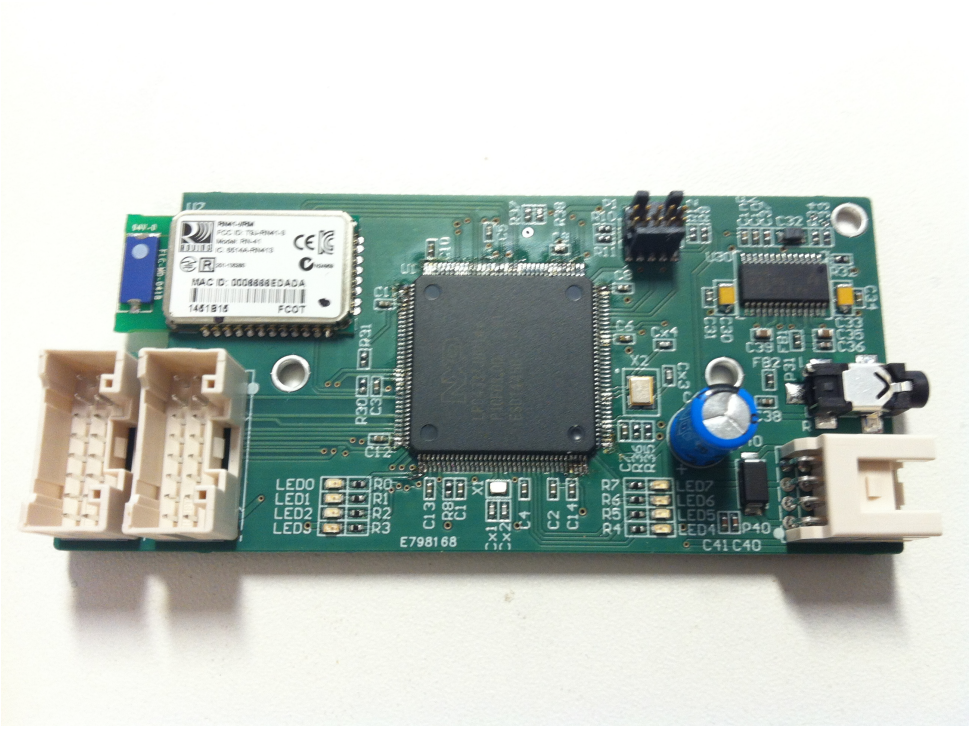


Figure 3.2: Controller board in KITT

The MCU is fed from the rechargeable batteries using a buck converter that converts the voltage to $5V$ DC. Using an LM1117-3.3, $3.3V$ 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, or the internal Bluetooth module of your laptop.

You can use your computer's internal Bluetooth connector to connect to KITT. If you want to use one of the lab computers or have trouble connecting to KITT from your PC (check if all drivers are installed), you can use the USB Bluetooth dongle.

KITT has four LED's, see figure 3.4, their meaning is as follows:

- Red (twice): $5V$ and $20V$ supply voltages are present
- Green (blinking): Bluetooth searches for connection; (steady) Bluetooth connected
- Yellow (blinking): Bluetooth data transfer

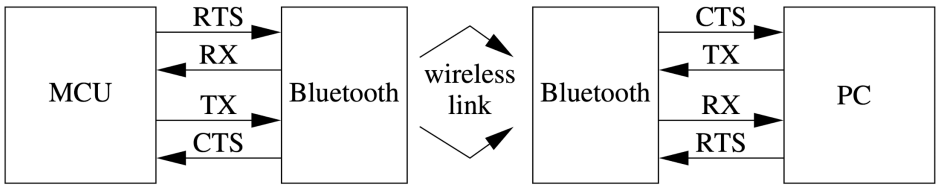


Figure 3.3: Schematic overview of the KITT communication with your PC

Figure 3.4: LED's on KITT

The button next to the LEDs is a reset button for the MCU. This reset button allows the user to reset KITT's actions if it freezes or is unresponsive.

3.3 COMMUNICATING WITH KITT

For this module, you will write several scripts fundamental for working with KITT for the rest of the project. For this, we will use the Python module pySerial to send and receive data to/from KITT.

3.3.1 pySerial

pySerial is a Python module that provides a simple and efficient way to communicate with serial ports. It allows Python programs to access and manipulate the serial ports on a computer, allowing them to communicate with other devices connected to those ports. With pySerial, you can easily send and receive data to/from these devices. pySerial is cross-platform, which works on Windows, macOS, and Linux operating systems.

Here are some of the basic commands used in pySerial:

- `serial.Serial(port, baud rate)` - This command initializes a serial connection. The port argument specifies the serial port to use (e.g., "COM1" on Windows or "/dev/rfcomm0" on Linux), and the baud-rate argument specifies the data rate in bits per second.
- `serial.write(data)` - This command sends data over the serial connection. The data argument is a bytes object that contains the data to be sent.
- `serial.read(size)` - This command reads a specified number of bytes from the serial connection. It blocks until the specified number of bytes is received.
- `serial.flush()` - This command is used to flush the input and output buffers of the serial connection.
- `serial.close()` - This command is used to close the serial connection.

These are just a few of the basic commands used in pySerial. Depending on the specific use case, many more commands and options are available. It is recommended to refer to the pySerial documentation for more detailed information: https://pyserial.readthedocs.io/en/latest/pyserial.html

3.3.2 Connecting to KITT

Before you can do anything with KITT, you must be able to connect to it. In this section we provide some basic commands to do this.

To start, connect the car to your computer over Bluetooth. Make sure you know the name of the Bluetooth port that the car is connected to. Next, to have access to this link in Python, use

```
serial_port = serial.Serial(comport, 115200, rtscts=True)
```

Notice that `serial_port` is an instance of pySerial. You can use this object to manipulate KITT. The variable comport is the path to the comport assigned to the Bluetooth connection with KITT. Which comport specifically KITT is connected to can be found in your communication settings.

E.g., if your transmission connection to KITT takes place over port 10, then you should use:

```
comport = 'COM10'
```

Connecting to a serial port over Bluetooth can be pretty tricky. Windows users, refer to Appendix B. Linux users can use `# rfcomm bind rfcomm0 MAC_ADDRESS`; there is also a section about it on the Arch Linux wiki https://wiki.archlinux.org/title/Bluetooth#Bluetooth_serial. The MAC address of KITT always ends with the code printed on its label. For example, if the label reads `DA:84`, the MAC will end with `xx:xx:xx:xx:DA:84`.

To find out which serial ports are available, you can use the `list_ports` tool of pySerial. Run the following in a shell: `python -m serial.tools.list_ports`.

The connection must also be ended when you are done working with KITT. This can be done by using:

```
serial_port.close()
```

The Bluetooth connection is disturbed by leaving comports open, quitting your work without closing the communication link, removing the Bluetooth dongle, or turning off the Bluetooth connection. Avoid disturbing the Bluetooth connection by ending the connection properly before doing any of the aforementioned things. If you disturb the Bluetooth link, you may need to reboot your computer to reset the operating system, which costs you valuable lab time.

3.3.3 Controlling KITT

Now that you can connect to KITT, you will write commands to control KITT.

Driving commands For driving, KITT supports two types of instructions: a direction command and a motor speed command. These control the motors using pulse width modulation (PWM). Both commands are neutral at a setting of `150`. The direction commands range from `200` (hard left) to `100` (hard right), and the motor commands range from `135` (backward) to `165` (forward). Note, however, that there is a dead zone for the motor commands, so KITT will likely not start moving forward until the PWM is set to about `153`. It is recommended to experiment with these values, they are also battery-dependent. You should test the size of the drive command dead-band, verify that 150 is the middle position for steering (sometimes there is a deviation of ± 2), and test the maximum left and right position.

All commands to KITT are sent in binary and should end with a new line character. The general format is:

```
serial_port.write(b'code\n')
```

The command used to set the direction to, for example, `169` is:

```
serial_port.write(b'D169\n')
```

The command you can use to set the motor speed to, for example, `135`, is:

```
serial_port.write(b'M135\n')
```

Note that once you set the motor speed commands, KITT will continue to act on this until you either transmit a new motor speed command or reset the MCU using the button on KITT, which will set both direction and motor speed commands to the neutral value of `150`. Do one of these two options after your tests since KITT will keep driving if you don't.

Audio beacon commands Later, you will also have to use the audio beacon atop KITT. This beacon can be turned on using:

```
serial_port.write(b'A1\n')
```

And turned off by using:

```
serial_port.write(b'A0\n')
```

Be aware that the default code word for the beacon is $0x00000000$, which means KITT will not start making noise on its own when the beacon is turned on. You should specify a code as described below before you can hear the beacon make noise. 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.

You can set the carrier frequency onto which a code is transmitted to a maximum frequency of 30 kHz. It can be set to, for example, 10000 Hz by using

```
carrier_frequency = 10000.to_bytes(2, byteorder='big')

serial_port.write(b'F' + carrier_frequency + b'\n')
```

Furthermore, you can set the bit-frequency of a code that is transmitted with OOK on the carrier frequency to, for example, 5000 Hz by using:

```
bit_frequency = 5000.to_bytes(2, byteorder='big')

serial_port.write(b'B' + bit_frequency + b'\n')
```

The repetition count, which sets the time between consecutive transmissions according to the formula $repetition\_count = bit\_frequency/repetition\_frequency$ and has a minimum of 32, can be set to, for example, 2500 by using:

```
repetition_count = 2500.to_bytes(2, byteorder='big')

serial_port.write(b'R' + repetition_count + b'\n')
```

The 32 bits code pattern is transmitted bit-wise over the beacon. This code must be specified in hexadecimal; say you use the hexadecimal $0xDEADBEEF$ as an example; the command to do this is:

```
code = 0xDEADBEEF.to_bytes(4, byteorder='big')
```

```
serial_port.write(b'C' + code + b'\n')
```

Status command Lastly, you can request a status report from KITT and receive all data from KITT by sending the request:

```
serial_port.write(b'S\n')
```

```
status = serial_port.read_until(b'\x04')
```

The status string reports the current drive commands, the ultrasonic sensor distance (cm), the battery voltage (mV), the audio status (on/off) and control parameters (code word, carrier-frequency, bit-frequency, repetition count). A sensor distance of 999 means overload (i.e., out of range)

Remember that all the numbers provided here serve only as an example. It is up to you to determine what carrier frequency, code word, etc., best fits the goal of succeeding at the final challenge.

3.4 IMPLEMENTING A BASIC KITT CONTROL SCRIPT

This assignment is part of the communication mid-term assessment. You are asked to write a Python program to control your car connected over Bluetooth. The program should allow the car to be controlled using the keyboard w, a, s, and d keys. The e and q should start and stop the audio beacon. Also add a key to stop communication with KITT.

The template script below defines a `class KITT` with essential methods for communication, and your task is to complete the script by implementing the `wasd` function.

```
import serial
import keyboard

class KITT:
    def __init__(self, port, baudrate=115200):
        self.serial = serial.Serial(port, baudrate, rtscts=True)
        # state variables such as speed, angle are defined here

    def send_command(self, command):
        self.serial.write(command.encode())

    def set_speed(self, speed):
        self.send_command(f'M{speed}\n')

    def set_angle(self, angle):
        self.send_command(f'D{angle}\n')
```

```
    def stop(self):
        self.set_speed(150)
        self.set_angle(150)

    def __del__(self):
        self.serial.close()

def wasd(kitt):
    # add your code

if __name__ == "__main__":
    # test code follows here
    kitt = KITT("your_port_here")
    wasd(kitt)
```

Below is a short explanation of the code. Ensure you understand how the code works before using its various functions.

if __name__ == "__main__"

- This is the code that will start executing when you run your script. This contstruct allows you to import your script as a module into another script without running the test code.
- An instance of KITT (an object of the KITT class) is then created, with the port as a parameter. You should change this to your port number.
- Next, the `wasd` function is called and given this instance of KITT. It should detect key-press events and attach these to KITT commands like
  - `kitt.set_speed(170)`
  - `kitt.set_angle(150)`
  - `kitt.stop`

KITT Class:

- The `__init__` method is the one that runs when you make an instance of the class. It starts the serial communication with the specified port and baud rate.
- `send_command` method sends a given command to KITT over the established serial connection.
- `set_speed` and `set_angle` methods adjust KITT's speed and steering angle, respectively.
- `stop` method brings KITT to a halt by setting both speed and angle to a neutral state.

- `__del__` method ensures the proper closure of the serial communication when the KITT object is deleted. It runs automatically at the end of the script.
- **TODO:** modify the `__init__` so that when the communication with KITT is started, the beacon is initialized with the correct set of parameters. Use the existing serial port and `send_command`.
- **TODO:** add two methods `start_beacon` and `stop_beacon` that turn the beacon on or off. Note that you should have set the beacon parameters during the `__init__`, so there is no need to resend them every time you turn the beacon on.

TODO: wasd Function:

- The wasd function is designed to be a continuous loop that reads keyboard events using the keyboard library which you loaded using `import keyboard`.
- When a key is pressed (`KEY_DOWN`), the function interprets the key and adjusts KITT's speed and steering angle or toggles the beacon accordingly.
- The 'w' key accelerates KITT forward, 's' stops KITT, 'a' turns KITT left, and 'd' turns KITT right.
- The 'e' key turns on the beacon, and the 'q' key turns off the beacon.
- When a key is released (`KEY_UP`), you could define appropriate actions, e.g. stop KITT or reset the steering angle.

3.4.1 Optional extensions

If you finish the basic assignment quickly and want to challenge yourself further, try adding additional functionality to the program. For example, you could:

- *Add a GUI:* You could create a graphical user interface (GUI) for the program to make it more user-friendly. The GUI could display information about the car, such as its speed or battery level, and provide buttons for connecting and controlling the car. You can use the built-in `tkinter` module but you are free to use any other GUI module, for example PyGame, pyglet, or PyQt.
- *Add error handling:* Currently, if there is an error with the Bluetooth connection or the serial communication, the program will simply crash. You could add some error handling to handle these cases more gracefully, such as printing an error message and exiting the program.
- *Add speed control:* Currently, the program only supports moving the car forward or backward and turning left or right. You could add support for controlling the speed of the car, such as by sending different commands to the car depending on how long the user holds down the forward or backward keys.

- *Add an emergency brake:* KITT doesn't have a brake; the `M150` speed setting lets KITT roll out to standstill, which might take a long distance. Add an emergency brake by letting KITT drive backwards for a short period of time. (You should first detect if the previous speed setting caused KITT to move forward. You need to define a state variable to memorize the speed setting.)

3.4.2 Mid-term assessment and report

After you finish this assignment, and ultimo in week 4, showcase the functionality of your script to your assigned TA. After you pass this assessment, you are ready to document your results in your midterm report.

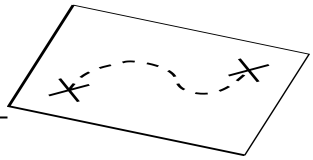
For this Module, you would include a chapter that covers the approach, implementation, testing and results of the basic controller; but postpone the documentation of a GUI to the final report. Review Chapter 7 for guidelines.

Remember to document your code, using comments to define input/output variables of functions and to explain the logic and any modifications made. Your completed script will be crucial for the upcoming challenges, contributing to the overall autonomous driving system.

After you have completed this module and have tested all the components thoroughly, start on the second part of the communication script outlined in Module 2.

Chapter ***4***

MODULE 2: READING KITT SENSOR DATA



Contents

4.1 OVERVIEW

KITT can only drive autonomously if it is able to sense its environment. Two types of sensors are provided: (i) two distance sensors mounted on the front of KITT, and (ii) five microphones located around the field which can record the audio transmitted by the beacon on top of KITT and send it to the PC.

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

- How to read out the distance sensors in front of KITT to detect obstacles
- How to read out the data from the microphones located around the field

Preparation

- Read this module
- Familiarize yourself with the datasheet of the SRF02 ultrasonic distance sensors provided on Brightspace.
- Windows 10 users: install ASIO4ALL and make sure you use a build of PyAudio that supports ASIO.
- Make sure your Python IDE is ready to go!

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- KITT
- Laptop/PC running Python
- Access to a field with a microphone setup and an audio card

4.2 THE DISTANCE SENSORS

KITT has two ultrasonic sensors on the front left and right. These ultrasonic distance sensors consist of the SRF02 module, the datasheet of this module is also on Brightspace. The module works by transmitting a pulse train at 40 kHz and then listening to its echo. The time until the first echo is received is measured and converted to a distance in centimeters. According to the SRF02 datasheet, the time between two observations (the cycle period) 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 sensors are started one after the other. The measurements are stored in a buffer on the MCU. A new measurement overwrites the older one.

To have a good understanding of the limitations of the system, you have to determine the working of the ultrasonic sensors. The practical realization of the SRF02 modules is simple and adequate for its purpose, which is for parking sensors. 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.

Moreover, the distance measured by the sensors must be relayed to the computer's control system. The various communication delays are at the origin of additional errors like KITT having already moved some distance before the new distance measurement reaches your PC.

Status commands The distance measurements are included in the output of the status request command, `serial_port.write(b'S\n')`, but an isolated version containing only the distances can be obtained by using:

```
serial_port.write(b'Sd\n')
```

Make sure to receive the data from KITT with the following:

```
serial_port.read_until(b'\x04').
```

Similarly, `b'Sv\n'` makes KITT send the voltage of the battery pack, and `b'V\n'` shows version information of KITT. Again, the data sent by KITT should be received with

```
serial_port.read_until(b'\x04').
```

4.3 DISTANCE SENSOR READING ASSIGNMENT

In Module 4, you develop a car model which includes dynamics (velocity, response to driving commands). To develop this, you will need plots of position versus time. When driving on a straight line

towards a wall, you can derive position from the parking sensor data. Further, in the final challenge, you will have to detect obstacles that could be in the path of KITT using the distance sensors. In this assignment, you should add sensor reading methods to the script from module 1. The received data should not just be stored in bulk but in a convenient format for later recall.

4.3.1 Assignment

Using the previously explained commands, have KITT transmit its various pieces of information. You can do this by adding a `read_sensors` method to the KITT class. Assign a key to show the distances measured. Display all the received data in an organized way in the terminal.

Task 1: Static measurements Perform measurements with the vehicle at stand-still. Introduce various obstacle configurations. Determine the accuracy of the sensors, their maximal range, and the field of view (beam angle).

Task 2: Delay estimates Make an estimate of the delays for getting an update on the distance, and figure out how they impact the performance of the control chain. The question is: how old is an estimate by the time you receive it? Some delays you can determine from the data sheets, and some delays you can measure using timing functions as described in Section 2.6.2.

Task 3: Dynamic measurements Perform distance measurements with KITT in motion (driving to the wall) and analyze the shape of the distance versus time. Is the plot continuous? Look at the plot for left and right sensor values, what do you see? Can you verify the 70 ms cycle time? Can you use these plots to estimate velocity accurately?

Task 4: Measurement data analysis and interpretation Analyze the measured data with an eye on the possibility of compensating for the possible errors. Implement some strategies to calibrate KITT. Include this data in your report.

4.3.2 Optional extension

It is advisable to store all the old distance data in a list inside the KITT class. This will be convenient during the final challenge, where the route planning might need old measurements to determine the position of objects. It will also help with characterizing the distance sensors.

A good way of doing this is by initializing an empty list in the `__init__` phase. Then, every time the sensor is read, you append this list with `[time, left_dist, right_dist]`.

4.3.3 Mid-term assessment 2.1 and report

After you finish this assignment, and ultimo in week 4, showcase the functionality of your script to your assigned TA. After you pass this assessment, you are ready to document your results in your midterm report.

For this Module, you would include a chapter that covers the above tasks (using independently-readable text, i.e., don't refer to "Task 1"). Include plots; for each plot it should be clear how the plot was made (i.e., the corresponding experimental set-up), and you have to describe what is seen in the plot before you discuss results and derive conclusions. Review Chapter 7 for guidelines.

Include the corresponding code in an Appendix. Remember to document your code, using comments to define input/output variables of functions and to explain the logic and any modifications made. Your completed script will be crucial for the upcoming challenges, contributing to the overall autonomous driving system.

4.4 THE MICROPHONES

Four microphones of the field are installed around its corners, and a fifth microphone is placed between two of the microphones at the edge of the field, at a level higher than the other four. These microphones, along with the beacon atop KITT, will be used to locate KITT within the field (chapter 5).

The appropriate sound card driver must be used to use the microphone array. The soundcard used in EPO-4 is a PreSonus AudioBox 1818VSL. On Linux, OSX (except the ARM version), and Windows 11, the sound card works out of the box. On Windows 10, it is necessary to install ASIO4ALL (https://www.asio4all.org/) and a build of PyAudio compiled with ASIO support (https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyaudio). If you use the Pipfile provided on Brightspace, the correct version of PyAudio should already be installed, but you still need to install ASIO4ALL manually.

A typical laptop will have many possible audio devices, for example the built-in microphone, a Bluetooth headset, and the AudioBox 1818VSL used in EPO-4. To initialize the microphone array, the correct audio device should be specified. This short script will list the index and names of all audio devices visible to PyAudio.

```
import pyaudio

pyaudio_handle = pyaudio.PyAudio()

for i in range(pyaudio_handle.get_device_count()):
    device_info = pyaudio_handle.get_device_info_by_index(i)
    print(i, device_info['name'])
```

On Windows 10, make sure to use the audio device index that has `ASIO` in the name. Other audio devices may be using the legacy `MME` or `WDM` Windows drivers, which may not support more than 2 synchronous audio channels.

The microphone array must first be initialized. When doing so, the sampling frequency that will be used must be specified. This sampling frequency will either be 48 kHz or 44.1 kHz, depending on the

type of audio device. Initializing the microphone array at device index `device_index` with a sampling frequency of `Fs` is done as:

```
stream = pyaudio_handle.open(input_device_index=device_index,
                             channels=5,
                             format=pyaudio.paInt16,
                             rate=Fs,
                             input=True)
```

To make a recording, the length of the recording must be specified. This must be specified as the number of audio frames to be recorded. The result will be a `bytes` object. Each audio frame will contain 5 samples, one for each microphone. Each sample contains 2 bytes, since we specified 16-bit audio. So, the return value of recording N frames is $10N$ bytes. To get a recording of `N` frames, one can run the following command:

```
samples = stream.read(N)
```

The following command turns this recording into a NumPy array:

```
import numpy as np
data = np.frombuffer(samples, dtype='int16')
```

At this point, the microphone data is interleaved: `data[0]` contains the first sample of microphone 0, `data[1]` contains the first sample of microphone 1, `data[2]` contains the first sample of microphone 2, and so on until `data[5]` contains the second sample of microphone 0. Table 4.1 explains the concept more visually. This interleaved data stream should be deinterleaved into 5 streams, one for each microphone.

Table 4.1: Recorded data from five microphones is interleaved into a single stream.

`data[0]`	`data[1]`	`data[2]`	`data[3]`	`data[4]`	`data[5]`	`data[6]`	`data[7]`	...
mic 0	mic 1	mic 2	mic 3	mic 4	mic 0	mic 1	mic 2	...
frame 0	frame 0	frame 0	frame 0	frame 0	frame 1	frame 1	frame 1	...

You may be familiar with the Matplotlib Python module, which can be used to plot the audio data received from the microphones. Matplotlib produces great-looking publication-ready figures. But drawing these plots can be slow. If you want to plot audio data in real time, consider using a more low-level GUI library such as PyGame or Pyglet and trade off beauty for speed. That said, Matplotlib supports some real-time features, such as animations. Just be aware that Matplotlib is not the only solution.

Lab rules regarding the microphone array Many groups will be using the same setup, and to avoid making other groups crazy, you are not allowed to rearrange the microphone connectors. Please also don't touch the volume settings. If these need to be adjusted, contact a TA.

4.5 MICROPHONE RECORDING ASSIGNMENT

The final part of communicating with KITT is using the 5 microphones. This will form an important part of the final challenge, a good implementation is thus essential.

Again, you should add a method like `record` into the KITT class, with an input N. This should turn on the beacon, make a recording of N sec, deconvolve the recording into its separate channels, store it for later processing, and turn off the beacon. Make a function to visualize the recordings; this will prove valuable in debugging.

4.5.1 Assignment

Do the following experiments and add the results to your mid-term report:

Task 1 Initialize the microphone array and record one of your team members clapping near the microphones one after another. Separate the data stream of each microphone from the interleaved data. Plot the data of all five channels, allowing you to identify which channel of your recording represents which microphone.

Task 2 Turn on KITT's beacon and record your results. Can you see the waveform of the transmission? Compare the waveform of the recording to an ideal OOK of your code. What do you see?

Task 3 Repeat the setup of **Task 2**, putting KITT nearer to one microphone than to others. Can you derive from the waveforms near which microphone KITT was placed? Show the plots you made and discuss your results and conclusions derived from them.

4.5.2 Optional extension

If you finish the basic assignment quickly and want to challenge yourself further, you could:

- See if you can automate selecting the correct PyAudio device index. The correct device index changes from one computer to another and can sometimes even change on the same computer after a reboot. So, it is worth your time to make a program that can automatically select the right device index.
- Implement start-up sanity checks: some process which you can run after you arrive at the test field, so that you can quickly check the microphone connections and audio levels.
- Explore PyAudio's callback mode. This manual describes what is called 'blocking mode'. The `stream.read()` function will block your program until the requested number of frames has been received from the sound card. You can instead specify a callback function to process new audio frames as they arrive. If done carefully, this will allow your program to respond faster to new microphone samples, and enable you to drive while recording. You can read more about callback mode in the official PyAudio documentation: https://people.csail.mit.edu/hubert/pyaudio/docs/

4.5.3 Mid-term assessment 2.2 and report

After you finish this assignment, and ultimo in week 4, showcase the functionality of your script to your assigned TA. After you pass this assessment, you are ready to document your results in your midterm report.

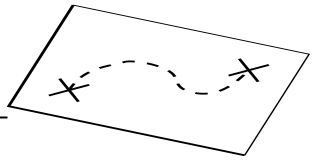
For this Module, you would include a chapter that covers the above tasks (using independently-readable text, i.e., don't refer to "Task 1"). Include plots; for each plot it should be clear how the plot was made (i.e., the corresponding experimental set-up), and you have to describe what is seen in the plot before you discuss results and derive any conclusions. Be sure to answer the questions posed along with the plots (using independently-readable text).

Include the corresponding code in an Appendix. Remember to document your code, using comments to define input/output variables of functions and to explain the logic and any modifications made. Your completed script will be crucial for the upcoming challenges, contributing to the overall autonomous driving system.

This concludes the mid-term assignments related to communication with KITT. After the mid-term, you must integrate this module with the localization module created by your colleagues. Take into account that integrating is often harder than originally anticipated, e.g. your code has to run in parallel, and you have to worry about timing aspects. Hopefully, using the KITT class will provide you with a sturdy and flexible framework to continue your work towards the final challenges.

Chapter 5

MODULE 3: LOCATING KITT USING AUDIO COMMUNICATION



Contents

In the final challenge, KITT must be located in its field and then directions must be determined to navigate to the final destination. In the previous modules your colleagues are developing scripts to communicate with KITT. They will add functionality to read the audio signals from the microphones located around the field, and you should use these to locate the car. It is recommended to read Modules 1 and 2 to have a better understanding of how everything should work together.

For the localization, we will use (real-time) recordings of the beacon signal at the various microphones, deconvolve these using a reference signal recording, and determine the relative time delays from the resulting channel estimates. (It is assumed that you have working channel estimation algorithms from the EE2T11 Telecommunications A course lab.) Depending on the distance to each microphone, the signal transmitted by KITT's beacon arrives a little bit earlier or later, and you can convert that into physical distances. For each pair of microphones, we can compute this time difference of arrival (TDOA), or the physical difference in propagation distance. If you have measurements from enough microphones, then you can calculate the location of KITT in the field.

To get you started, on Brightspace you can find 7 recordings with known locations, and a reference recording taken close to one microphone. These can be used to develop and test your algorithms.

On Brighspace, you will also find 3 recordings without the location given. These will be used to assess the effectiveness of your algorithm. You will have to compute the location of these recordings live in front of your assigned TA during week 3. It is not necessary to use real-time recordings for the mid-term challenge. Make a detailed report of your experiments and all steps involved for the mid-term report. More information can be found in Chapter 9.

At the end of this Module, you will have developed a script to locate KITT within the field with reasonable accuracy, and you will do so using the data recorded by the microphones located along the field. You will also have tested and verified the accuracy and robustness of your solution.

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

- The importance of good auto-correlation properties of your bit-code
- Channel estimation and TDOA from recorded signals
- Location estimation algorithms based on TDOA

Preparation

- Read this module
- Review your TDOA and deconvolution algorithms from the EE2T11 Telecommunications A practical
- Study Appendix C
- Make sure your Python IDE is ready to go!

Time duration Four lab sessions and four preparation/reporting sessions at home

What is needed

- KITT
- Laptop/PC running Python
- Your TDOA and deconvolution algorithms from EE2T11 Telecommunications A
- On Brightspace: 7+3 pre-recorded audio recordings.

5.1 PRE-RECORDED DATA

On Brightspace, you can find 7 given recordings. These recordings have locations randomly distributed across the field. Use them for testing and the mid-term challenge. The location of the recordings is mentioned in the file name. For example, the first recording is called "record_x64_y40.wav".

Table 5.1: Locations of the given recordings (cm)

x	y
64	40
82	399
109	76
143	296
150	185
178	439
232	275

The x and y axes are defined as follows, where the numbers refer to the microphone index:

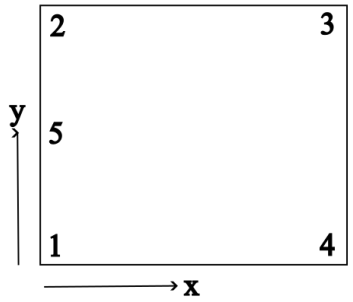


Figure 5.1: Microphone axis definition

TA's will setup the fields at the beginning of each session, you can assume these positions for the microphones. Please note the different height of microphone 5.

Table 5.2: Location of the microphones (cm)

Microphone	x	y	z
1	0	0	50
2	0	480	50
3	480	480	50
4	480	0	50
5	0	240	80

5.2 BACKGROUND KNOWLEDGE

In the EE2T11 Telecommunication A course lab, you developed algorithms for channel estimation for 2 microphone signals, both receiving a beacon (training) signal. In the present module, we extend this to 5 microphones and use this to locate the car.

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:

ch1 Deconvolution in the time domain: Involves matrix inversion. It is computationally complex and requires lots of memory (easily more than what the available PCs can handle).

ch2 The matched filter: Avoids the matrix inversion. It is equal to computing the cross-correlation of $y[n]$ with $x[n]$. As cross-correlation is equivalent to a convolution with a reverse signal $x[-n]$,

it can be computed efficiently using the FFT. The resulting channel estimate is equal to the true channel convolved with the autocorrelation of the pulse, $x[n] * x[-n]$. Therefore, its performance heavily depends on having the correct (i.e., accurate) "reference" or "training" signal to correlate your measurements with, and the signal should have good autocorrelation properties: $x[n] * x[-n]$ should be close to a delta spike. Large sidelobes will lead to confusion.

ch3 Deconvolution in the frequency domain: Involves the FFT. It computes the same channel as via deconvolution in the time domain but is much more efficient: convolution in time becomes pointwise multiplication in frequency, hence for deconvolution in the 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). Since this method is similar to ch1 (but much more efficient), you can view this method as a matched filter, followed by a correction step that "inverts" the effect of the autocorrelation of the transmit pulse. However, since we should not divide by zero, 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 has to be chosen heuristically.

It is assumed that you have working algorithms for estimating channel responses. You developed these during the EE2T11 Telecommunication A practicum. We recommend using ch3: deconvolution in the frequency domain.

In each case, a reference signal $x[n]$ is required. We recommend using a recording close to the beacon because then $x[n]$ includes the loudspeaker and microphone responses as well. For the pre-made recordings, a high-quality reference is available.

The deconvolution algorithm gives a channel estimate for the beacon path to each microphone. After this, we detect the first incoming path, which, assuming Line of Sight (LOS), corresponds to the propagation delay of the car beacon to each microphone. Unfortunately, we do not know the transmit time, so we only obtain relative propagation delays. If we take the difference of microphone delays, this unknown transmit time is eliminated, so we can obtain time difference of arrival (TDOA) samples for each microphone pair. In the next sections, we will use these to locate the car.

5.3 DECONVOLUTION

The first step is to get your deconvolution algorithm operational and tested. Do these tasks with the given recordings from Brightspace.

Task 1 Using the reference signal provided and the algorithms you developed at the EE2T11 Telecommunications A practical, deconvolve the recordings to get the channel impulse response for each microphone. You will need to segment the received data into individual pulses; for the moment, you can do that "by hand".

For a few measured locations, plot examples of the segmented data and the deconvolved channels (e.g., 10 plots per recording: the inputs and outputs of your deconvolution algorithm).

Task 2 From the peaks in the channel estimates, determine the time of arrivals (TOAs) and store these in a table. Check the accuracies of these estimates. *Hint:* Obviously, you want to compare these estimates to your true TOAs. The problem is that the transmission time is unknown, so a direct comparison is impossible. You can either compare TDOAs (the pairwise differences of TOAs) or introduce a single unknown parameter (the time of transmission) and develop an error measure that is insensitive to it. This is harder, but you might find literature on this.

Task 3 Once you are satisfied with the performance of the basic algorithm, extend your code with ways to automatically segment the received data and find the beginning of a pulse. *Hint:* A useful function is `findpeaks`, which allows you to implement criteria for finding the first strong peak in a pulse sequence.

Deliverable In the midterm report, document your algorithms, show examples of measurements/channel estimates, and include a subsection on testing, showing your findings and accuracies.

As plots, we suggest to show an entire recording, and then one where you zoom in on the short segment that you give to ch3. Of the resulting channel estimate, show the entire result, and then zoom in on the interesting part where are the peaks. Remember that for negative delays, the peak of interest will be at the "large n" part of your estimate.

5.4 LOCALIZATION USING TDOA INFORMATION

Now we arrive at the main question studied in this module: how can we locate the car using the TDOA estimates? With 5 microphones, we can compute the TDOAs between all pairs of microphones and obtain 10 TDOA pairs.

Study Appendix C: it shows a basic algorithm to compute the (x,y) location of the car based on the measured TDOAs. This algorithm can be extended to fit 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 the microphones are placed on stands at a different height. You can assume that the height difference between the audio beacon of the car and the microphones is known.

5.4.1 Localization class

Now, it is time to start setting up a processing pipeline. The localization class you will develop will take a 5-channel recording of input and return the x and y coordinates of the car. Below is an outline of a localization class you could use. Note that this is to help you get started, and you will have to add methods and logic yourself. Look at the comments inside of the class for an explanation.

```
class Localization:
```

```
    def __init__(self, recording, debug=False):
        # Store the recordings
        # Load the reference signal from memory
        # x_car, y_car = self.localization()

    def localization(self):
        # Split each recording into individual pulses
        # Calculate TDOA between different microphone pairs
        # Run the coordinate_2d using the calculated TDOAs

    def TDOA(self, rec1, rec2):
        # Calculate channel estimation of each recording using ch2 or ch3
        # Calculate TDOA between two recordings based on peaks
        # in the channel estimate

    @staticmethod
    def ch3(x, y):
        # Channel estimation

    def coordinate_2d(self, D12, D13, D14):
        # Calculate 2D coordinates based on TDOA measurements
        # using the linear algebra given before

if __name__ == "__main__":
    # Main block for testing
    # Read the .wav file
    # Localize the sound source
    # Present the results
```

5.4.2 Localization algorithm assignments

Task 1 Add your favorite channel estimation function from the Telecommunications Lab. Test this by calculating some channel estimates of the given recordings.

Task 2 Develop a test code using Pythagoras' theorem that takes an (x, y) position as input and calculates the TDOA that you would observe from this position. This involves calculating the distance from the given point to each microphone. This will help you debug both your TDOA function and your `coordinate_2d` function.

Task 3 Complete the TDOA pipeline. It should return the time difference between all pairs of microphones. Compare with results calculated by the test code from task 2.

Task 4 Implement the `coordinate_2d` method and test the module individually before connecting it to other modules. *Hint:* If you know the true location, you can calculate the error. But since the TDOA vector is redundant, your function from Task 2 can also be used to generate an error estimate, based on your estimated position, even if you do not know your true position.

Deliverable Document the accuracy of your localization algorithm. Is this acceptable for your application? As an illustration, you can also plot the room and show the true locations and estimated locations. For the midterm report, document your algorithm and the results of the tests on simulated and given test data. Also report on the given test data with unknown position: what positions do you estimate? Finish with a conclusion that summarizes the accuracy that can be expected and the reliability (i.e., how often it succeeds in finding a reliable location).

5.4.3 Mid-term assessment 3 and report

In week 4, you will have to showcase the functionality of your localization script to your assigned TA. You should demonstrate proper localization of the car on the 3 recordings with unknown coordinates.

After you pass this assessment, you are ready to document your results in your midterm report. A detailed report is required, covering the approach, implementation, testing and results, as mentioned above. Please review Chapter 7 for guidelines.

5.4.4 Optional extensions

If you finish the basic assignment quickly and want to challenge yourself further, try adding additional functionality to the program. For example, you could look at one of the following aspects:

- The current set of linear equations in Appendix C does not consider the height difference between the microphones and the car. This leads to a slight offset. Adding in the 5th microphone at a different height can fix this. Use logic and maths to adjust the matrices to accommodate this extra microphone and the z-axis.
- The provided method in Appendix C 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 nonlinear Least Squares optimization toolboxes.

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.

- Try a different channel estimation function.
- Apply filters to detect outliers and average the result of multiple pulses.

If you have completed this successfully, you can start integrating and estimating the car's location. Start by finding a set of suitable parameters for your audio beacon. Communicate with the other part of your team. They should be finalizing the microphone recording code. If this is not the case, work together to complete the code. It is better to complete the mid-term first with well-tested code and then to continue working on more functionality. Otherwise, you will spend all your time trying to integrate all the loose bits of code in the end.

5.5 AFTER THE MIDTERM: REFERENCE SIGNAL AND INTEGRATION

As you will remember from your EE2T11 Telecommunications A practical in Q3, you must have a clean reference signal to get a good channel/impulse response estimate, which you can then use to deconvolve the recordings you make to locate KITT. Getting this clean reference is what you will do here since it entails more than just simulating the OOK code, as you must also consider the microphone's channel and the beacon's behavior.

This reference is crucial in finding the channel's impulse response between the beacon and the microphone and, consequently, in finding the TDOA to locate KITT within the field.

Appendix A reminds you of the beacon signal parameters that are used to generate the beacon signal; they are similar to what you saw in the EE2T11 practical.

5.5.1 Reference signal assignment

Task 1 Determine a good bit code to transmit using KITT's beacon, for this consider its autocorrelation function. You want a strong peak for the 0-lag of the autocorrelation but as low as possible for any other lag.

Hint: Suitable codes could be randomly generated, or you can try some optimal codes (check communication theory literature for "gold codes").

Task 2 Determine the other parameters for KITT. What is a good carrier frequency to use? What is a good bit frequency? A perfect repetition count is not yet required as this will depend on your later script but do make sure that the full code can be transmitted and that it can be recorded on all microphones within the same recording window.

Hint: You could try to find a datasheet of the microphones to determine a good carrier frequency; it depends on the sensitivity of the loudspeaker and the microphones. More reliably: you can measure the response of the system for various carrier frequencies (e.g., spaced by 1000 Hz) and make a plot of the amplitude response. You can then later select a carrier frequency with maximal response.

Task 3 Record the bit-code transmitted over KITT's beacon with your new parameters. Keep the microphone very close to KITT's beacon to get a recording that is as clear as possible, but do make sure to avoid clipping. Look at the waveform you get from this recording. Can you think of a way to reduce the noise? If you cannot develop a way to do this, ask a TA or other instructors.

Task 4 The recording you made with the microphone close to the beacon can be the reference signal. Look at the waveform of the reference and look at its autocorrelation. What do you see? Compare this to the "ideal" OOK and autocorrelation from **Task 1**.

Task 5 Clean up your recording and strip "zero intervals" away such that only a clean recording of a single pulse remains.

Deliverable Show the autocorrelations and plots you made in the final report and comment on what you see. Explain your answers to the questions, what led you to those answers, and explain your choices.

5.5.2 Integration KITT assignments

Task 1 Record several transmissions from KITT's beacon with KITT at various locations in the field. Store these recordings and KITT's (x, y)-coordinates.

Task 2 Using the reference signal you developed previously and the algorithms you developed for the mid-term, estimate the locations using these recordings, off-line.

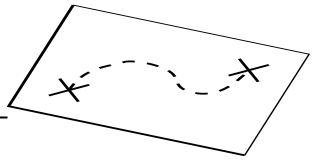
Task 3 Do the same as in Task 2, but now couple the location algorithm and the recording code. Assuming you solve the "blocking recording" issue, you should now be able to drive KITT around and locate it all in real-time.

Task 4 Try to add time-stamp information. By the time you calculated your position, KITT has already moved. And note that in your recording, you look for a pulse, which is also from some time ago. Luckily, you can estimate most of these delays. Augmenting your location estimates with time stamps might be very helpful for the controller that your team will build next, in particular if you intend to drive fast.

Deliverable Document this process and reflect on your findings and accuracies.

Chapter **6**

MODULE 4: CAR MODEL



Contents

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

The model has two parts: modeling the velocity, and modeling the direction. This directly relates to the two controls you have: velocity setting and steering angle. The overall system is a non-linear "non-holonomic system", meaning it is hard to directly apply your control knowledge to this system. But decoupling the system into velocity and direction is a good way to approximately linearize the system.

The idea of deriving a good model is that it improves your understanding of the system. Also, it allows to test your control algorithm first off-line on your model, and if proven right, going to the real KITT should be less painful.

Learning objectives System modeling in action: Physical model for steering a car.

Deliverable As part of the mid-term report: A brief discussion on the assignments, and a tested system model.

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

- Reading this Module
- Homework Tasks 1 and 2 in these sections.

Time duration Three lab sessions, three homework sessions.

6.1 VELOCITY MODEL

We start by driving on a straight 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:

Figure 6.1: Indicate the forces and their directions.

- 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.

  Unfortunately, KITT does not have a brake! You can stop by letting KITT roll to standstill, or for a short period apply a negative force F_b. The difference with a real brake is that if you apply F_b for too long, or were already stopped, the car will drive backwards.
- Force F_d due to the air drag and other types of friction. This opposing force also decelerates the car and depends on the velocity v as

  $$F_d = b|v| + cv^2 .$$

  The absolute value is needed to ensure the friction opposes the driving direction.

For the moment, 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

The car's state parameters are longitudinal position z (not to confuse this with (x,y) later), and corresponding velocity $v = z'$ and acceleration $a = v'$, all functions of time.

Homework Task 1

1. Indicate the forces and their directions in Figure 6.1.
2. Write the differential equation describing the longitudinal car dynamics.

3. Check if the air drag is relevant for our setting. Assuming it is not, you can easily solve the first order differential equation in closed form. Derive this solution!

4. Make acceleration and deceleration plots, both position z and velocity v as function of time. Use a small step size, e.g., 0.01 second. Set the initial velocity $v(0)$ to 0 meters per second. Simulate the model for 8 seconds using the maximum acceleration force $F_{a,\max}$. Do the same for deceleration, starting from an initial velocity $v(0)$, and applying $F_{b,\max}$.

   Interpret the results. Do they correspond to what you would expect from an accelerating/decelerating car?

Hint: You can also use Simulink to create such a model.

Velocity assignments

Our aim is to take our theoretical model, and tune it to the real KITT, by estimating the relevant parameters (i.e., *model identification*). As it is a simple model, it is not going to be perfect. The dynamics of the car are non-linear, e.g., it takes some extra force to make the car start driving. Also in practice battery conditions do not remain constant, the car dynamics are then time-varying. We will try to ignore these effects.

Further, the motor settings range from 135 till 165, with 150 being "neutral"; you can assume that 165 corresponds to $F_{\max}$, and 135 to $-F_{\max}$. So you will need a translation table from input to force, that hopefully is linear.

The control input is the speed setting that we transmit to KITT. The system state is the position and velocity (derivative of position). The observations are the distance of KITT to the wall, as measured with the distance sensors. Measure elapsed time using a Python timer, see Section 2.6.2.

1. Make sure the batteries remain at a constant voltage during this experiment. Record the battery voltage at each experiment.

2. Start/stop at different speed settings (applied during fixed time intervals) and generate the associated position-time curves.

3. Consider how to determine velocity from such plots.

   Hint: this is not as easy as it seems. Obviously you think of finite differences, $(z(t)-z(t-T))/T$. This will give you the velocity not at time t but at an earlier time. Also, if you sample fast, the inaccuracy of position has a great effect on the accuracy of the velocity estimate. If you sample slow, you obtain velocity at a significantly earlier time.

4. How quickly do you reach a constant velocity? (This depends on the speed setting.)

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

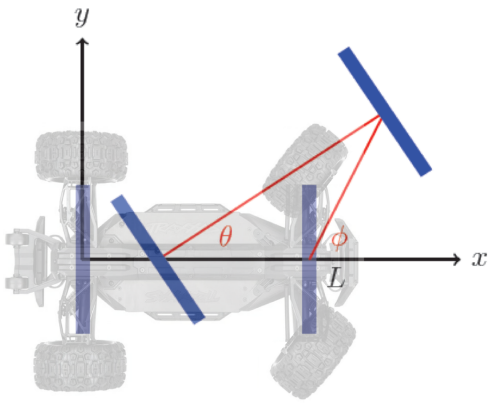


Figure 6.2: Approximation of KITT's movement

6. If you drive backwards, does the same model fit?

6.2 STEERING MODEL

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 from the previous section 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

$$\begin{bmatrix} v\cos(\phi)\,\Delta t \\ 0 \end{bmatrix}$$

and the front axis at

$$\begin{bmatrix} L + v\cos(\phi)\Delta t \\ v\sin(\phi)\Delta t \end{bmatrix}.$$

The situation is sketched in Figure 6.2.

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

$$\begin{aligned}\frac{d}{dt}\theta(t) &= \lim_{\Delta t\to 0}\frac{\theta(t+\Delta t)-\theta(t)}{\Delta t}\\ &= \lim_{\Delta t\to 0}\frac{\arctan\left[v\sin(\phi)\Delta t/L\right]}{\Delta t}\\ &= \lim_{\Delta t\to 0}\left(1+\frac{v^2\sin(\phi)\Delta t^2}{L^2}\right)^{-1}\frac{v\sin(\phi)}{L}\\ &= \frac{v\sin(\phi)}{L}.\end{aligned} \tag{6.1}$$

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

Finally, instead of working with angles, it is often easier to work with unit direction vectors, e.g.,

$$\mathbf{d}(\alpha) = \begin{bmatrix}\cos(\alpha)\\ \sin(\alpha)\end{bmatrix}.$$

where α is the orientation of the car ($\alpha = 0$ if the car is parallel to the positive x-axis). The reason it is easier is that we don't have to worry about 2π wrapping if we increase or decrease α by some amount. If we have a certain direction vector $\mathbf{d}$ and we want to rotate it over an angle θ, we can use a rotation matrix

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

and set $\mathbf{d}_1 = \mathbf{R}(\theta)\mathbf{d}_0$. Similarly, we can define an orthogonal direction vector

$$\mathbf{d}(\alpha)^{\perp} = \begin{bmatrix}-\sin(\alpha)\\ \cos(\alpha)\end{bmatrix}$$

that points into the direction of the center of the driving circle (you may need to use $-\mathbf{d}^{\perp}$).

The state vector of the car consists of position $\mathbf{x}$ and orientation α, or equivalently unit direction vector $\mathbf{d}(\alpha)$, and a velocity v. You have to consider here what point of the car you consider to be the reference point. The control input is the steering angle ϕ, or equivalently the direction setting given to KITT.

Steering assignments

- Assume a constant velocity. Derive an equation for the radius R of the circle as function of steering angle ϕ. Show the derivation in your report.
- Is this equation also valid in case the velocity is not constant?

- The wheelbase L is 33.5 cm (see Figure 3.1). Choose a direction setting and drive KITT in a circle. Measure R and derive the corresponding steering angle ϕ.

  Repeat for different direction settings and create a calibration table that relates the direction setting to steering angle.

- Make a more detailed plot than Figure 6.2, showing also the center of the circle, $\mathbf{c}$. Which part of the car (back wheels, center, front wheels) actually rotates around $\mathbf{c}$? If the car is at position $\mathbf{x}$ and the direction vector of the car is $\mathbf{d}(\alpha)$, then what are the coordinates of $\mathbf{c}$?

  Hint: Use $\mathbf{d}(\alpha)^{\perp}$.

- You now have all the ingredients to create a dynamic model, given the initial state $(\mathbf{x}_0, \mathbf{d}_0, v)$ and a steering angle ϕ, compute the next state after a time Δt.

  Hint: You also have to define to what part of the car $\mathbf{x}$ refers to. Presumably, this is the location of the beacon.

6.3 COMBINED MODEL

As the final step, combine the steering model with the earlier velocity model to also update v over the same period Δt.

We recommend to create a `class KITTmodel`, with a very similar interface as the earlier `class KITT`, i.e., it accepts the same driving and steering commands. The virtual model can update the state parameters accordingly, each time the state is requested and each time the speed or steering setting is changed. You will need an internal timer to track how much time has elapsed since the previous time the state was updated (i.e., Δt, which is not a constant).

This is more advanced OOP, and tricky to debug due to the real-time aspects. Develop this first for just the velocity part and a one-dimensional distance z.

Alternatively, you can opt for a simpler approach where you develop a simple script that carries out a series of commands, and records the resulting positions.

6.3.1 Mid-term assessment 4 and report

In the midterm assessment, the TA will give you a short series of driving/steering commands (e.g., M165, D150 for 0.5 sec, D170 for 0.5 sec, stop), your model predicts the position, and this is compared to the actual position of KITT after following the same series of commands.

In the midterm report, present the model, and present test results that shows the accuracy of this model, starting from a known state. Summarize in a conclusion: over what time period can you predict the new position of the car with an accuracy better than 30 cm?

6.3.2 After the Midterm: GUI integration

After the midterm, you will work on a controller that takes you from *A* to *B*. Part of this is to have a GUI where you show the field and the measured position of the car.

If you have a good car model, you can develop the controller and test it on this model. You can show the resulting estimated positions of the car in the GUI. This is convenient for debugging, since your testing time on the real field is very limited.

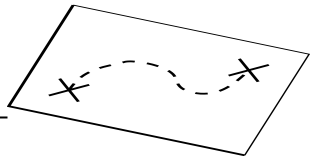
If your model is reliable, then at a later stage you can try to *fuse* location estimates from the beacon signals with location predictions from the model. For optimal fusion, you need to know the accuracy (variance) of each of the estimates, so you can take a weighted average. You can also use the model predictions to detect and reject possible outliers from the beacon method.

You will probably find that the car behavior is not constant; e.g., the velocity response depends on the battery status and also (slightly) on the steering setting. You could extend your model to take that into account.

Further, to make the virtual model interface similar to the actual car interface, you would have to integrate the Module 3 Localization scripts in such a way in the `class KITT` that you can request the current state (position, orientation) from the real car just as you request it from the virtual model.

Chapter 7

MID-TERM REPORT



Contents

Now that you know how to drive your car, and hopefully have a working localization module and car model, it is time to document what you did so far in a mid-term report. This report will also form the basis of your final report.

Report A mid-term report explaining primarily the localization and modeling results. Also answer the specific questions asked in the modules. Suggested length: about 15 to at most 20 pages (excluding the Appendix that lists the Python code).

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

Preparation Completed the preceding Modules (sign-off by the TAs).

Time duration Two homework sessions.

7.1 MID-TERM REPORT

The mid-term report explains your designs with its technical details. The primary focus is to report on your results for the localization and modeling modules, but also the distance sensor measurements should be reported. Suggested length: 15 to at most 20 pages.

Aspects on which the report is judged are:

- *Technical content:* 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; penalty for unacceptable conceptual mistakes or unfinished work.

  Proof of your results (localization algorithm, car model) using sufficiently extensive testing.

- *Quality of the submitted report:* Conformity to the requirements of a scientific report (adequate use of equations, figures, citations, cross-references), readability, layout.

  The format is of that of a technical report, and not that of a homework assignment. Thus, the report should contain the results of the various assignments, embedded in a natural way, as these will motivate your solution.

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

The deadline for submission is listed on Brightspace. Submit your report using the corresponding submission folder. Please use a filename that starts with your group number.

The report should be independently readable by a technically skilled committee member who is not familiar with EPO-4, but you can refer where needed, so don't copy large parts of this manual but summarize the scope in your own words. Note: you cannot refer to something like 'ch3' without explaining briefly what that is.

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 concise, and judge yourself what is important to be included.

7.2 HOW TO WRITE AND STRUCTURE YOUR REPORT

Here are some general directions on the structure and contents of your report (mid-term and final report). For the mid-term report, the main focus will be on three sections: sensor data, localization, and car model; other chapters such as system integration and conclusions are not yet relevant and can be omitted.

Keep a clear structure and writing style. Make sure you give sufficient factual information (in particular if things don't work). The language should *not* be informal. Keep things concise, "bla-bla" is not appreciated. If you make claims such as "something is the best", first provide evidence (or at least a motivation, or a reference to literature).

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.

A picture (block scheme) might be helpful to show the structure and the relations among subsystems. Sometimes, after the introduction a section is needed that describes the background in more technical or mathematical detail, or analyzes the problem in more detail. For the mid-term report, that structure is probably overkill.

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), but that is probably not needed for EPO-4.

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 (i.e., don't include a text on the relevance of autonomous driving). ChatGPT-generated text is not appreciated.

Sections: You can probably skip to report on Module 1, but place the results of the other Modules each 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").

For each section, think of the following aspects:

- *Specifications:* What is the objective? What is given already? 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.
  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.
- *The resulting design:* This could comment on the implementation, refer to the main variables that you use, etc; generally after reading this, a reader should be able to quickly grasp the Python code in the Appendix.
- **Testing/verification:** How do you test your solution is functional and meets the specifications? What did you measure/observe? Present your results (e.g., measurement results, a Python 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/evidence and discuss what is seen).

For EPO-4, particular emphasis will be placed on the testing aspect. You cannot make claims unless you provide evidence. Your overall system will fail if a subsystem fails, and the purpose of testing is to rule out possible causes of failure.

– *Conclusions:* each section finishes with a conclusion summarizing the results of the Module in a few lines, including claims on expected accuracy. This captures what members of the other sub-group need to know when they use your Module as a black box tool.

System integration: This will be added in the final report. Apart from the integration aspects, this is a natural place to describe a GUI, and its functionality. Include a screenshot.

Also in this section, you need a subsection on Verification, now on the complete design (overall test). The results of the final challenge may be added here as well.

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.

Discussion: Here you can include information about the (group) process and any other useful feedback. This part can also include your original planning (work division over team members and time) and the actual outcome. See the Kickoff document: how did that work out? Did everyone contribute?

The material under Discussion 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 Python code and/or other details on your design that may be helpful.

– Structure your code into functions.

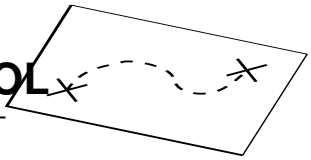
– Functions should have a header block that explains what the function does and what the input/output parameters are. Also include author and date information (version, history).

– Describe data structures and other global variables in sufficient detail.

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

Chapter 8

MODULE 5: STATE TRACKING AND CONTROL



Contents

This chapter contains information that you will need when preparing for the second part: the final challenge. The ultimate goal is to let KITT drive from an initial (known) location *A* to a given target location *B*. 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 in this chapter on how you could attack the problem (but many alternative solution approaches exist).

Learning objectives Basics of system engineering. Extension of control theory knowledge.

Deliverable Sections of your final report regarding the problem analysis and overall design, and the more detailed design of the control system.

Python code for control that was tested on your virtual car model

Preparation Read the chapter. This module provides ideas you may use.

What is needed Python code that implements your virtual car model.

Time duration One session for brainstorming (high-level system design). Two sessions for developing the control system in Python 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 DEFINING YOUR APPROACH

At the beginning of this second part, it merits to brainstorm with your entire group on the approach that you will take. For your selected approach, determine whether it will do the job (and under what

assumptions/conditions). You can opt for a simple and robust "combinatorial" approach (i.e., based on many if-then-else statements), that will be able to handle the expected situations but not more than that, or a more thoughtful approach based on control theory, that is much more robust but also more risky to implement.

This high-level system design is part of *system engineering*, i.e., with the entire system in mind, define an approach at a high level that will do the job, and define specifications of each constituting subsystem. These parts can then be designed by specialists that don't have to know or consider the entire system. If the parts are tested and verified to meet their specifications, then the entire system is supposed to work. (Or so you would think.)

Deliverable

In your final report, document your selected approach, and the alternatives that you considered, plus your motivation for the selection.

Document the foreseen consequences of this choice. Draw a block scheme that shows what software blocks should interact, and define the interaction (e.g., variables).

Your solution will probably contain a finite state machine and/or a loop. An important consideration is the timing of this loop. Document your analysis of this in sufficient detail. How often do you intend to measure your location, keeping in mind the constraints and trade-offs here. Consider also the various delays in the system. If you obtain a location fix, how old is that information? If you estimate velocity from two locations, then to what time point does that velocity refer? How fast should one iteration of the loop be? (If it is too fast, then you won't have new location information, but if it is too slow, then you might miss your target.) Can you merge location fixes from the audio beacon with predictions of your position using your car model?

The text on this in your final report can be placed into an initial section "Problem definition and analysis", or "Problem analysis and high-level design", depending on how you want to organize your report. Alternatively, it can be placed after the localization and car modeling sections, if you need to use information from those sections.

8.2 DEFINING A TARGET PATH TO THE GOAL

Although not necessary, it is helpful to create a target trajectory from A to B. Assume that you know your starting position and orientation (i.e., the initial state). From here, you can draw a feasible trajectory to the target B in many ways.

- You have some freedom in your orientation when you arrive at B.
- Since steering amounts to driving on circles, you can define a circle with the smallest possible radius and then drive straight, or define a circle with a larger radius that goes through both given points and is tangential to the given orientation, or anything in between.

- In some cases you may need to drive backwards to be able to reach the goal. E.g., what happens if the target is within the smallest turning circle? [Note: This scenario is not part of the basic challenges, but you could use it for the Free Challenge. See Chapter 9.]

- Depending on your control strategy, you may need to recreate the target trajectory each time new location information is measured.

Deliverable

In the final report, document your path planning solution. Illustrate this with examples of generated routes under varying conditions.

8.3 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. In one dimension and with a known position, velocity and braking curve, it is quickly clear how we can find the moment to brake. In two-dimensional space, this is much harder: this is called a *non-holonomic system*.

8.3.1 Partitioning by projection

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 8.1 depicts this idea. The line represents the trajectory that we would want to follow.

Let KITT's trajectory be given by

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

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 we travel on S. KITT's speed should then be continuously controlled so that z approaches the desired distance without overshooting. Thus using z as a measure for distance allows for one-dimensional control in a plane.

This approach requires to know KITT's trajectory: we need an *estimate* of KITT's future movement. This is given by the planned trajectory. We can now 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 velocity control along any trajectory in a plane.

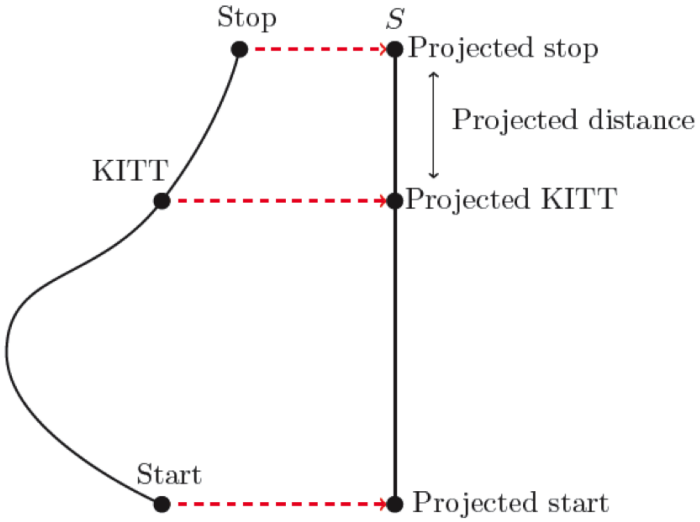


Figure 8.1: KITT's trajectory

8.3.2 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.

Intuitively, you will think of a solution where you know your current position/orientation, and you always steer towards the target. Once you are oriented towards the target, your "angle error" is zero, and you just have to drive straight. In all other cases, the angle error determines how much you need to steer. It is easy to see that this approach might work, but also might become unstable once you are very close to the target.

Suppose KITT is driving on its trajectory S, where its orientation 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 orientation. If KITT's orientation 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 (6.1) yields the autonomous non-linear system

$$L\frac{d}{dt}\theta + v\sin(k(\theta - \theta_t)) = 0\,. \tag{8.1}$$

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

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

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

$$\frac{d}{dt}T(\theta) = -(\theta - \theta_t)\frac{v\sin(k(\theta - \theta_t))}{L}. \tag{8.2}$$

Figure 8.2 depicts both $T(\theta)$ and its time-derivative.

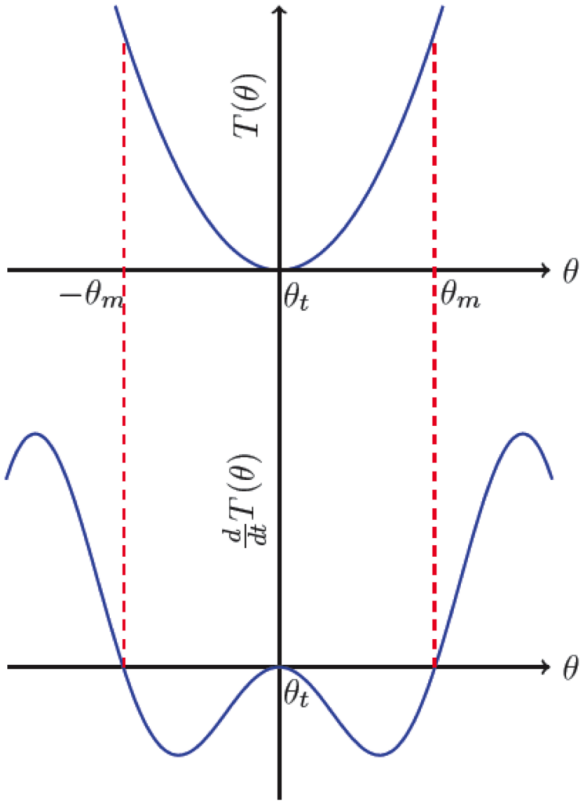


Figure 8.2: Graph of the potential function and its derivative

Consider KITT's orientation at any instant. If $|\theta| < \theta_m$, then by Figure 8.2 the potential function will have a negative derivative. But then it will decrease over time, and 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 (8.2) we can now state that Equation (8.1) 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.

8.3.3 Deliverable

Implement your control algorithm and test it using your virtual car model. In your final report, document the results of the tests, which prove that given correct position estimation, the control algorithm will get the car to its goal.

8.4 OPTIONAL: 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 (your velocity and steering commands), $\mathbf{x}$ is the (true) state vector, and $\mathbf{y}_k$ is an observation, in our case your position estimator. For example, for moving in 1D on a line, without friction, and with a discretization time step ΔT,

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

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). The fundamental idea here is that we do not have access to the true state $\mathbf{x}_k$, we only have estimates $\hat{\mathbf{x}}_k$ that we keep updated.

If the observation noise $\mathbf{v}_k$ is high, then we don't trust the measurements $\mathbf{y}_k$, and will estimate the next state as $\hat{\mathbf{x}}_k = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_k$, i.e., using the model predictions only. On the other hand, if the input (state) noise $\mathbf{w}_k$ is high, we don't trust the model state update and estimate the state (position, velocity) mostly from the observations. The Kalman filter gives a weighted average of these two extremes, using something called the Kalman gain. Our trust is modeled by covariance matrices $\mathbf{Q}_w = \mathrm{E}[\mathbf{w}_k\mathbf{w}_k^T]$ and $\mathbf{Q}_v = \mathrm{E}[\mathbf{v}_k\mathbf{v}_k^T]$. For example, $\mathbf{Q}_w$ contains on its main diagonal the measurement noise variance on your position estimates, which you documented in the preceding Modules. $\mathbf{Q}_v$ is the jitter on your input, and you could measure this by repeatedly giving the same control inputs and seeing how the state varies (measuring the state with a ruler rather than the less accurate beacon locator).

Kalman filters are described in many books, see e.g. Chapter 7.4 in

> Monson H. Hayes, *Statistical digital signal processing and modeling*, John Wiley and Sons, New York, 1996. ISBN: 0-471 59431-8.

Unfortunately, steering a car is highly nonlinear. You could try to use an Extended Kalman filter. Alternatively, since you have 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!

8.5 OBSTACLE AVOIDANCE

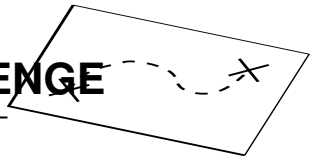
Obstacle avoidance is an advanced topic and you will only get to this if you got trajectory tracking completely solved and working.

You have the parking sensors to help you detect obstacles, and also the perimeter of the field can be considered an (invisible) obstacle. Once you detect an obstacle, you need to steer around it. Some suggested approaches are:

- Define a new planned trajectory around the obstacle (possibly requiring driving backwards). Typically, students follow a *combinatorial approach*: lots of if-then-else statements. This works in simple cases. The disadvantage is that it is hard to debug, and the solutions are often not general.
- Study control literature on obstacle avoidance that use *artificial potential fields*. This is a general approach that essentially defines a penalty function around obstacles and then finds optimal trajectories that minimize the "cost".

Chapter 9

SYSTEM INTEGRATION AND 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 yet to come: making it work all together.

During the final challenges, you and your team must navigate KITT from its starting location to an end location by combining what you've learned in the previous modules. You will need to use your locating algorithms in combination with a control system to drive KITT along a path to get it where it needs to go. You will also use the distance sensors on the front of KITT to avoid obstacles.

At the end of this module, you will have written a script that combines the modules and allows you to complete the challenges (hopefully as the fastest team).

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

– Combining all modules into one well-functioning control system

Preparation

– Read this module

– Ensure your code from previous Modules is tested and working

What is needed

– KITT

– Laptop/PC running Python

– Access to a field with a microphone setup and an audio card

– Your locating algorithm from Chapter 5

– Your car model from Chapter 6

– Your control algorithm from Chapter 8

– Your script to make distance measurements with KITT in motion from Chapter 4.

9.1 SYSTEM INTEGRATION

Before you can start with the integration part, it is necessary to have completed all the previous subsystems. To link these parts together, you must communicate with the complete group about the responsibilities. The assignment during system integration is to complete the big final design such that you can complete the challenges. Read this chapter completely to know what they are.

Focus on the basic challenges first. Once these have been implemented, you can work on the obstacle avoidance and anti-collision system.

Take it step-by-step, and keep it simple and structured. You'll quickly reach a level where the car shows unexpected behavior and nobody understands why.

For the control system, you can choose to plan a route but you are not required to do so. Important to note is that the location estimates from the locating algorithm must be taken into account during the challenge as the integration of these two is the central focus of the system.

When integrating the control system with the locating algorithm, it is important keep in mind the time it takes to run your control loop after each locating attempt and consider how you can ensure that you always record a full beacon waveform. Ideally, you drive your car while estimating locations. If necessary, it is permitted to stop the car while measuring. This is much easier to program, but you certainly won't be competing for speed that way.

After you have completed the integration of the control and locating algorithm, you can attempt an implementation of the obstacle detection and avoidance system for challenges C and D.

If you have time, we recommend to design a GUI; see Figure 9.1 for an example. This will facilitate to enter parameters for each challenge, and help you to keep an overview on what's going on. But if you have a tight control loop, you'll have to watch out that updating the graphics is not taking too much time.

Since you have a virtual car model, you could also use a switch and have the virtual car drive the track, so that you can test the control and obstacle detection performance on those moments that the real field is not available.

Do not postpone to document your work. The final report is needed very shortly after the demonstrations.

9.2 FINAL CHALLENGE

The final challenge for EPO-4 consists of four challenges and a fifth "free challenge". In order to get a passing grade, you have to successfully pass at least the first challenge. With every other challenge you

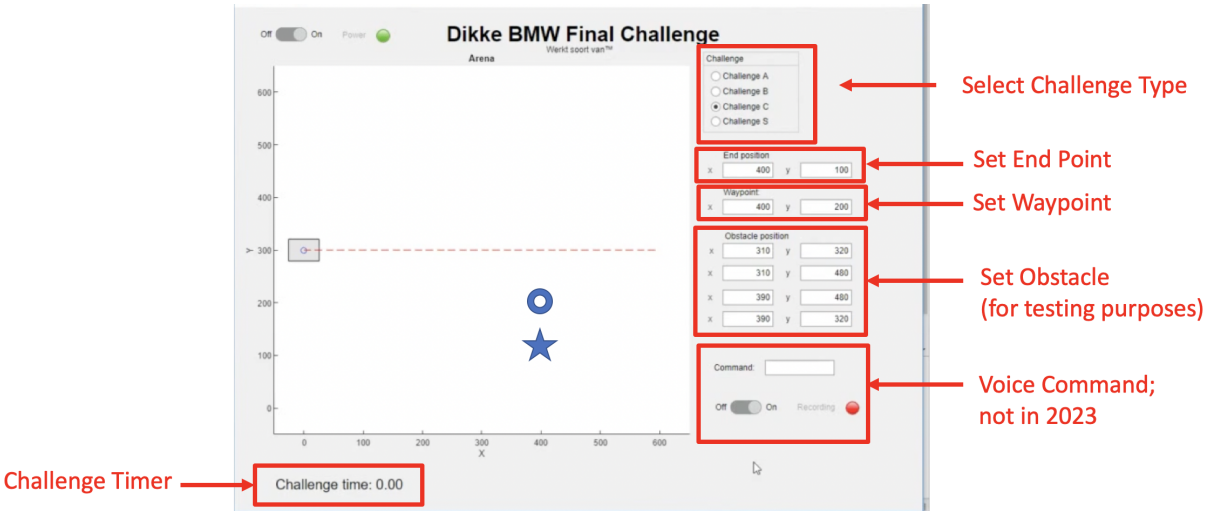


Figure 9.1: Example Graphical User Interface

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".

The car reference position is the beacon location. Before each challenge you are allowed to measure the field and the position of the destination (and waypoints). A challenge starts at a given (known) starting position at the edge of the field. The orientation of KITT is always 90° with respect to the edge 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. Needless to say, you must use the audio beacon for locating KITT, and the ultrasonic sensors for obstacle detection (no "open loop" solutions allowed).

Challenge A (60 points)

- KITT drives from the starting position to a specific point A in the field.
- Once KITT reaches the destination it must stop and the PC must give a signal.

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) the examiner must have have time to measure the distance: you should let the car stop for 10 seconds.
- When KITT reaches the destination (A) it must stop and the PC must give a signal.

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 be remembered.

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.
- There is an obstacle on route (two paper bins on top of each other) that 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 after B you will need to drive backwards to make the turn.
- 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[1].
- 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 and for every additional 10 cm you will lose 2.5 points, this holds for both the way point and the destination.
- For each time KITT hits an 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.[2]

[1] The grading for a task cannot be negative.

[2] It is not possible to receive more than 100 points for the whole competition.

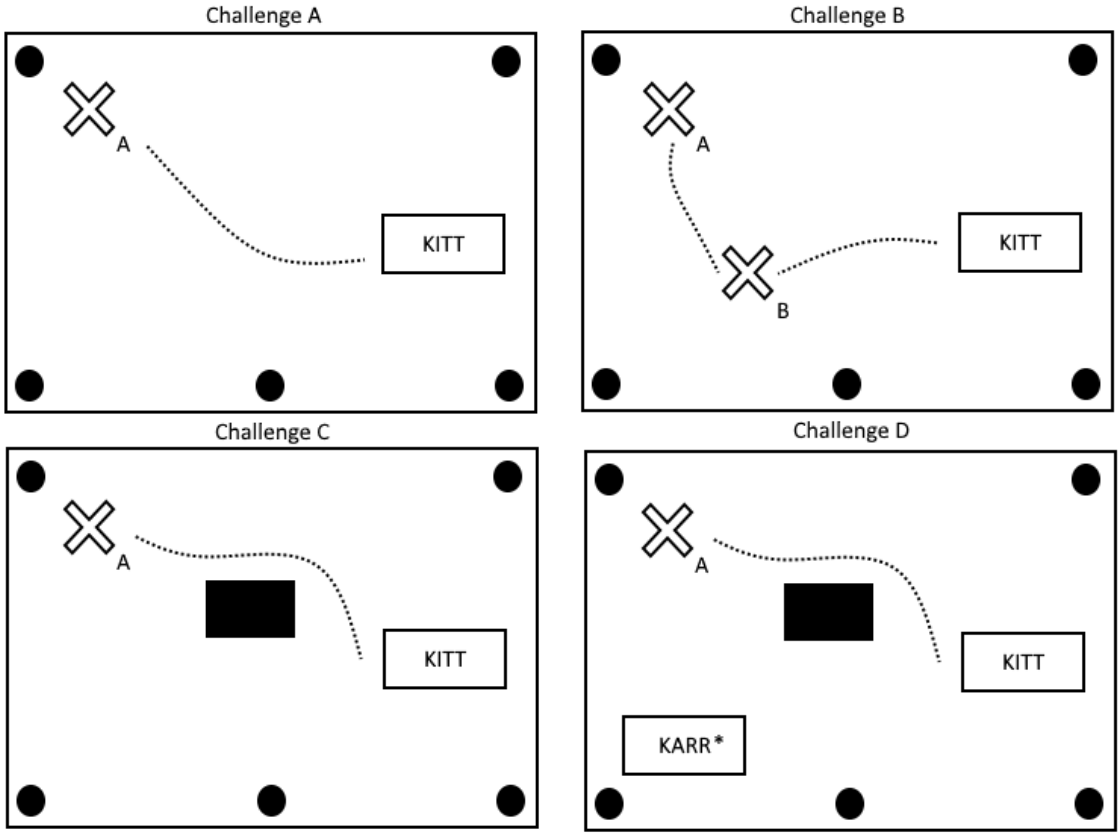


Figure 9.2: Example depiction of the Challenges. (*On the old television series, KARR is the archenemy of KITT)

9.3 FINAL REPORT

Instructions for the final report are similar to those of the midterm report (see Chapter 7). Aim for a **well-structured**, compact yet complete report of about 30 pages (plus Python code in an appendix). Do not forget to systematically 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. Note that the final challenge is not a test, rather it is a demonstration. With extensive testing, these results won't be a surprise to the reader!

The report is judged by committee members that are not indepth familiar with EPO-4, or the manual. Your report has to be sufficiently self-contained.

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

9.4 FINAL PRESENTATION AND DISCUSSION

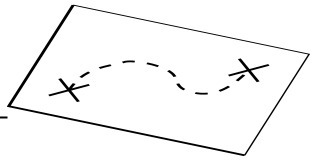
In week 10 (consult Brightspace for the exact date), you present and defend your final report in front of an examination committee. The examinators will ask questions about your design choices and aspects of teamwork. This will be part of your grade.

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.

Appendix ***A***

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 generated signal is similar to the audio signals you have used in the EE2T11 Telecommunication A practicum (Labday 4). The parameters that define the signal can be set through the command interface; in contrast to the earlier practicum, now it is possible to use an arbitrary carrier frequency, bit frequency, and repetition count.

A.1 AUDIO BEACON SIGNAL PARAMETERS

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

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

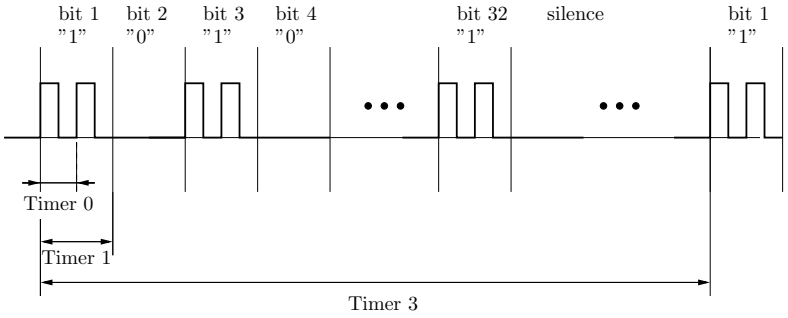


Figure A.1: An example of the pulses generated by the audio beacon

For example, with a value of 5 kHz of the bit frequency, 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 by sending them to the microcontroller on the car over the Bluetooth interface. The corresponding commands are described in Section 3.3.3.

The model `refsignal` that is used in EE2T11 Telecommunications A practicum has similar parameters, but were limited to a specific set of values.

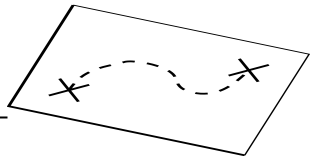
The default setting of the audio beacon is a 32 bit code sequence bit-stream with:

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

There is no guarantee at all that this default setting is of high quality!

Appendix ***B***

SERIAL COMMUNICATION WITH WINDOWS



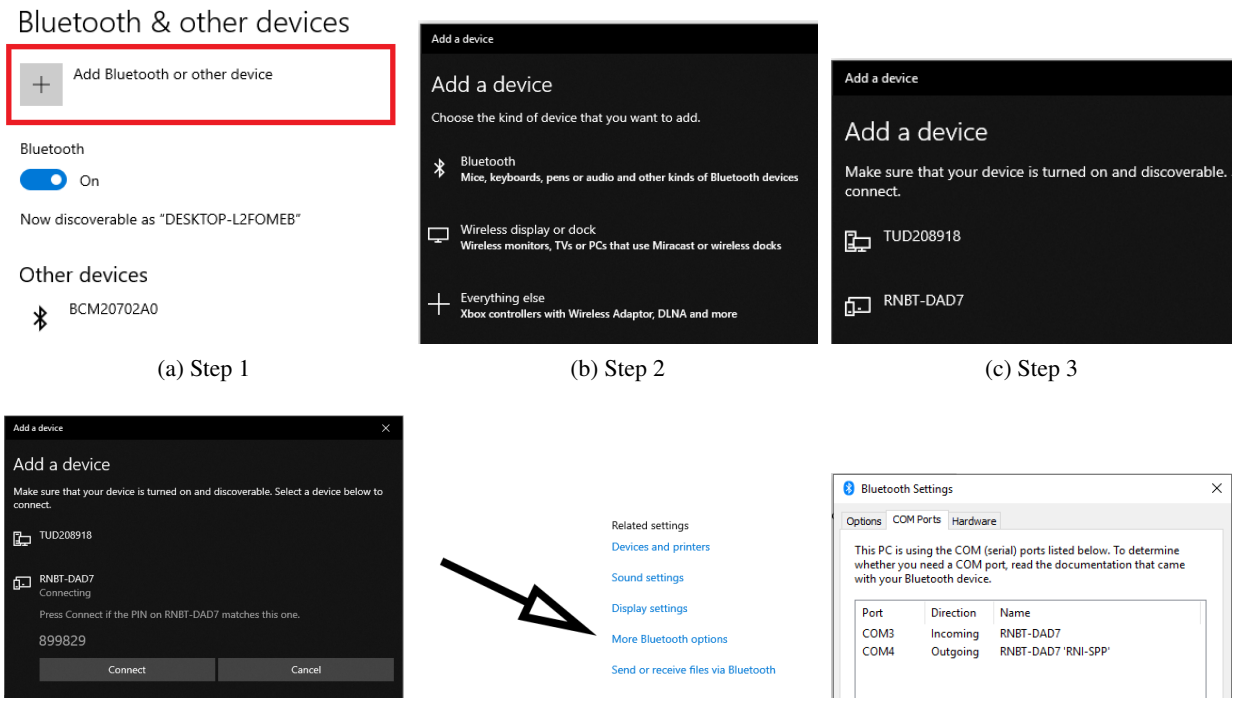
Contents

This Appendix describes how the serial connection to the car from Windows (using Bluetooth) is implemented.

B.1 CONNECTING KITT TO A PC ON WINDOWS 10

In Figure B.1 the steps can be seen how KITT can be connected to a PC with Windows 10 installed.

- Note that all KITTs have their Bluetooth hardware address written in the front of the car. Use this identifier to choose the right Bluetooth device to pair: Steps 3 and 4 (see Figures B.1c and B.1d).
- In Step 6 (see Fig. B.1f), the correct port to take note of is the one listed as `Outgoing`. This is the port you have to use in your setup.



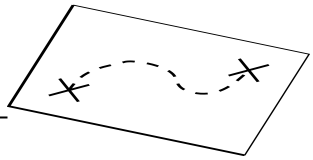
(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 4 (e) Step 5 (f) Step 6

Figure B.1: Steps how to connect KITT to the PC

Appendix **C**

TDOA LOCALIZATION ALGORITHM



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

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

We will use

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

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

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

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 a matrix equation as

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

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., $\mathbf{y} = \mathbf{A}^\dagger\mathbf{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$.

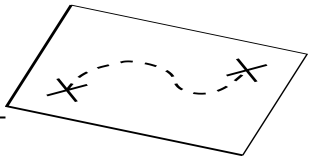
We thus obtain $\mathbf{x}$, as well as the nuisance parameters d_2, d_3, d_4 that depend on $\mathbf{x}$.

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

Question: what happens if $r_{12} = 0$?

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 EPO-4 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 EPO-4 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. In the context of EPO-4, a sprint of 1–2 labdays is probably appropriate.

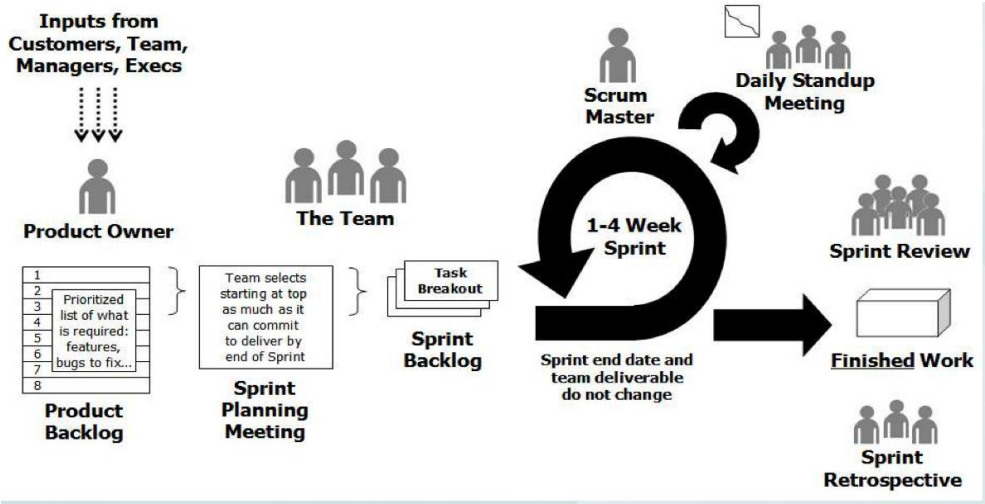


Figure D.1: Overview scrum development method[1]

D.2 TASK ASSIGNMENT

1. *Product owner*

   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 EPO-4, the product backlog is the list of requirements for 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.

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 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 EPO-4, 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] https://www.linkedin.com/pulse/20141021025927-21583419-mobile-development-using-agile-scrum. Last checked on 16 April 2019.

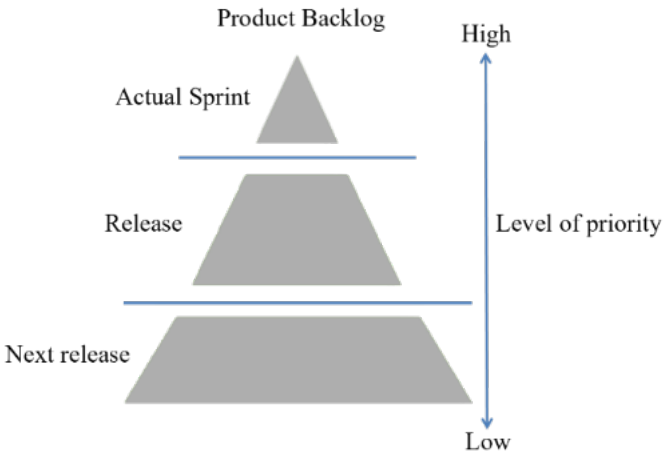


Figure D.2: Product backlog divides user stories in 3 categories [2]

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. *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*

[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.

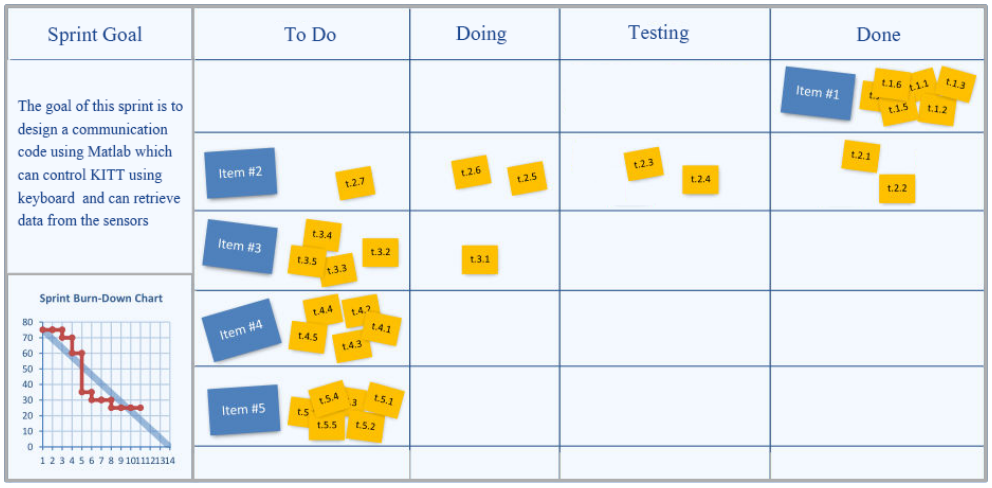


Figure D.3: Sprint planning and burn-down chart[3]

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.

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.

[3]Edited from: http://mayakron.altervista.org/wikibase/show.php?id=Scrum.

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.

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 EPO-4 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.