Ackerman-like Autonomous Vehicle

Islam Mohamed 1[1], Abdallah Mohamed 2[1], Naier Nabil 3[1], Yasmine 4[1], Rawan 5[1]

***Abstract*—In this project an ackerman vehicle steering mechanism was implemented on autonomous vehicle. Simulation using ROS Melodic and Gazebo was carried out to develop multiple nodes for speed and lateral control of the vehicle, and also for the localization problem using Kalman Filter to clear the noise. The Hardware was implemented using ROS enabled Raspberry Pi 4 and 2 Arduinos for Measurement and Control Purpose. Gyroscope and Optical Encoders were used as the sensory elements.**

***Index Terms*—**

I. Introduction

An autonomous car is one that can sense its surroundings and operate without the need for human intervention. At no point is a human passenger required to assume control of the car, nor is a human passenger required to be present in the vehicle at all. An autonomous car can go anywhere a traditional car can go and perform everything a skilled human driver can do. The difference in turn radius between the front tyres is known as Ackerman. On oval track automobiles, it is sometimes beneficial to have the left front tyre turn faster than the right front tyre. Ackermann steering geometry is a geometric arrangement of connections in an automobile or other vehicle's steering system designed to handle the problem of wheels from the inside and outside of a curve needing to track approximately circles of differing radii. Ackerman steering is a type of steering that is extensively employed in vehicles to improve handling. The steering system has evolved over time to include electrical and hydraulic steering systems, but the underlying idea remains the same.

II. State of Art

The combination of an MPC and a PID control is used in [1] to offer a technique for lane maintaining and longitudinal speed control of an autonomous vehicle. The suggested control approach aims to reduce lateral deviation and relative yaw angle in relation to the planned trajectory while maintaining the highest possible longitudinal speed. The reference profile of longitudinal speed is calculated using the vehicle's lateral and longitudinal dynamics. A linear 3-DoF bicycle model is used to describe the vehicle. The sole external input used by the control algorithm is the road lane borders. Three different driving scenarios are used to validate the proposed technique in simulation.

Obstacle avoidance and overtaking are critical skills for autonomous vehicles. The avoidance of a collision with a car, a passenger, or any other obstacle, as well as the production of a possible continuous curvature trajectory, are the key challenges that researchers confront in developing a safe path planning system. In [2], According to Pengwei Wang et al.'s research, to create an obstacle avoidance path planning approach, first create a safety model of obstacle avoidance by analyzing the obstacle avoidance behavior of a human driver. The artificial potential field method is then refined based on the safety model, and the repulsive field range of obstacles is recreated. Finally, a collision-free path for autonomous driving cars is produced using the improved artificial potential field. Co-simulation and real-world vehicle tests are used to verify the suggested algorithm's performance. The created path satisfies the restrictions of roadways, dynamics, and kinematics, according to the results.

The majority of self-driving car control frameworks are built on a middle-ware layer that consists of numerous separate modules linked by an inter-process communication mechanism. By subscribing and reporting events about their state, these modules implement fundamental activities and report events about their state.In [3], hierarchical interpreted binary Petri nets (PNs) were introduced to define the behavior expected from the car in different scenarios according to the traffic rules. Modules can communicate with each other using messages to update the internal states.Also, A programming environment named RoboGraph (RG) is established to use the middle-ware inter-process communication mechanisms to coordinate the actions in the rest of the modules to execute a behavior where the modules are distributed into 4 categories. First of all is The hardware driver includes all the processes that manage the hardware devices on board the vehicle (sensors and actuators).Secondly, The control layer implements the basic navigation functions: reactive control or obstacle avoidance (local navigator), localization (localization), path planning (map manager), and perception (event monitor) that process information from sensors to detect the basic events and followed by The executive layer coordinates the sequence of actions that need to be executed by other modules to carry out the current behavior. Finally ,The interface consists of a set of processes to interact with the users and the web.

III. Simulation

In this project ROS (Robot Operating System) is used for simulation purposes.
ROS is a comprehensive operating system for service robotics, similar to operating systems for PCs, servers, or independent devices. ROS is a meta-operating system that sits somewhere in the middle of an operating system and middleware. It's a collection of open source software libraries that allow users to create robotics applications for free.
It provides a range of features standard to an operating system:

*This work was not supported by any organization

[1]Mechatronics Department - Faculty of Engineering and Materials Science - German University in Cairo, Egypt
mail1, mail2, mail3, mail4, mail5

- Hardware abstraction
- Contention management
- Process management

ROS also provides high-level functionality:

- Asynchronous calls
- Synchronous calls
- Centralized database of data
- Robot configuration system

Before robot computer systems, each robot developer and robotics researcher might devote a significant amount of time to developing both the embedded software and the hardware of a robot. Mechanical engineering, electronics, and embedded programming were all required. Typically, the programmes developed in this manner were more analogous to embedded programming, which is similar to electronics, than to robotics in the truest sense, as we might see it today in service robotics. Because programmes were so closely linked to the hardware resources, there was a lot of re-use.

The main idea behind a robotics operating system is to avoid constantly reinventing the wheel by providing standardised functionalities that perform hardware abstraction, much like a traditional operating system for PCs, hence the similar name.

ROS is a language-agnostic operating system. At this time, three main libraries for ROS have been defined, allowing ROS to be programmed in Python, Lisp, or C++. Aside from these three libraries, two experimental libraries are available, allowing ROS to be programmed in Java or Lua.

A. Selection of Model

Ackerman Model was used for simulation in Gazebo which is a ROS enabled physical simulator.
The Ackermann model selected was from an open source avaiable model called (ackermann vehicle)

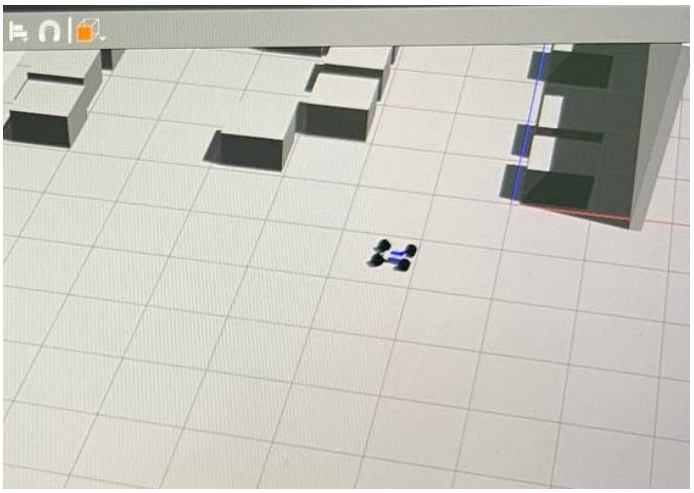


Fig. 1: Ackerman Model in Gazebo

B. Open Loop Response

The used model was tested in an Open Loop Manner to explore the actuation and sensing implemented in the model to be able to use it.

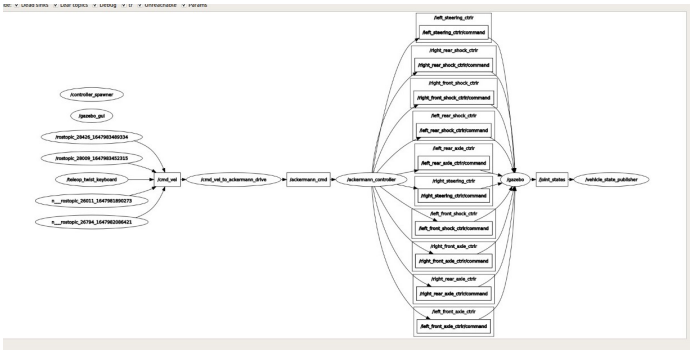


Fig. 2: Open Loop Response

C. Teleoperation Driving Mode

Furthermore, the selected model was tested in an Open Loop manner using teleoperation ROS node which was implemented to enable controlling the vehicle using the keyboard

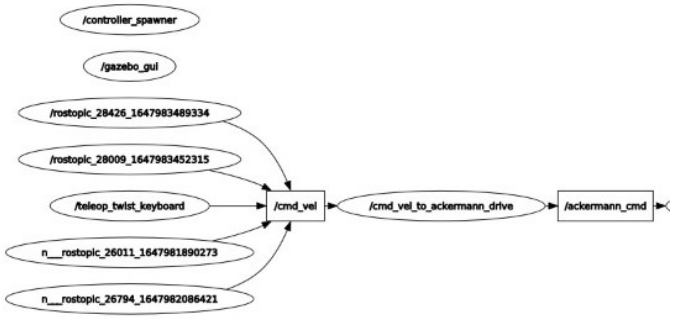


Fig. 3: Open Loop Response

D. Teleoperation

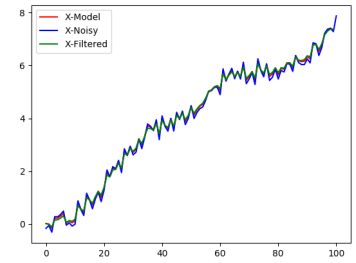
Two ROS nodes were implemented for Speed and Lateral control of the robot respectively.

The Speed Control Node uses PID Control to control the speed of the model.

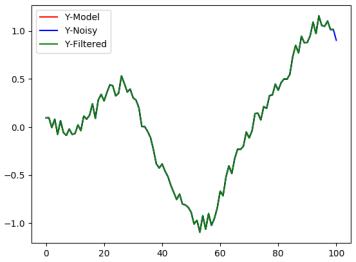
The Lateral Control Node uses Stanely/ Puresuit Control for lateral distance control.

E. Localization

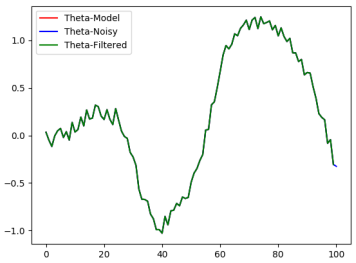
For the sack of simulating a real world scenario:
Localization Node was implemented to extract the states of the model and add some disturbance to them. Then using Kalman Filter these noise can be filtered.



(a) X



(b) Y



(c) Theta

Fig. 4: Different Vehicle States

F. ROS Model Architecture

The ROS Publishing and Subscription states between the multiple nodes are shown here

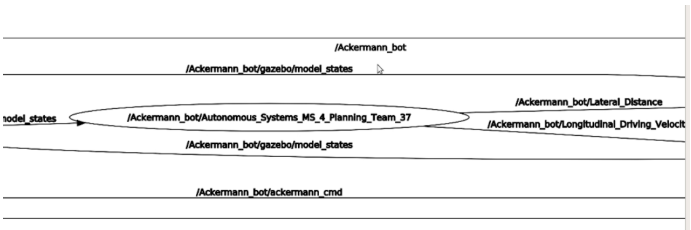


Fig. 5: ROS Model Architecture

Where Planning ROS Node publishes the desired speed and lateral distance to ROS Speed Control Node and ROS Lateral Control Node respectively. Where ROS Control Node subscribes to topics from these two nodes to publish the actuation signals to the model.

Then the Localization node publishes model state topics to :

- ROS Speed Control Node
- ROS Lateral Control Node
- ROS Control Node

IV. Hardware

The project hardware objectives can be divided into 5 main steps :

1. Purchase of the components and the suitable car chassis
2. Rasperry Pi Environment Setup 3. Installation and implementation of the optical encoder
4. Installation and implementation of the gyroscope sensor
5. Fixation and integration of the final hardware circuit and soldering the wiring together
6. Hardware testing and adjusting the Arduino code for lateral and longitudinal control
7. Hardware Communication between Rasperry Pi and Arduino

A. Rasperry Pi 4

Raspberry Pi is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. Because of its minimal price, modularity, and open design, it is widely used in many fields, including weather monitoring. Due to its implementation of the HDMI and USB standards, it is primarily used by electronic and computer hobbyists.

ROS Melodic was installed on the Raspberry Pi to trasnfer all the developed ROS nodes in the simulation on the Hardware.

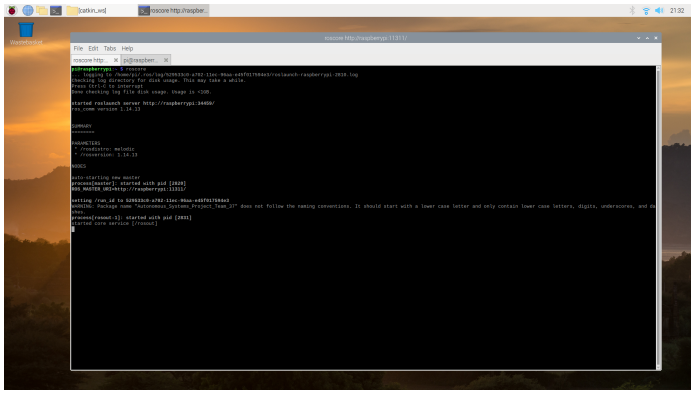


Fig. 6: ROS Melodic on the Raspberry Pi

1- Telepoperation ROS Node was implmented on the Rasperry Pi and the Hardware Vehicle.

2- Control Nodes were implemented on the Hardware using the Raspberry Pi too.

3- Localization Node was implemented on the Raspberry and the Hardware Vehicle to read the values from the sensors

and print them on the screen to monitor them.

4- SSH Protocol was enabled to be able to control the Raspberry Pi remotely while being in action.

5- Communication between the Raspberry Pi and one Arduino for the reading of the sensory data.

6- Another Communication was implemented between the Raspberry Pi and other Arduino for the control signals.

Fig. 7: Raspberry Pi Communication

B. Optical encoder (IM393)

The optical infrared encoder has a very simple function. It comes with an encoder disk which contain 20 slot which is fixed on a rotating shaft (rear shaft) and between the transmitter and the receiver towers so that every single revolution outputs 20 high-to-low clock signals and by simple calculation the motor speed can be detected by counting the number of revolutions per min and multiplying it by the wheel radius (v = w*r) using simple Arduino code . As shown in the pictures above the encoder has a simple pin diagram consisting of 3 pins a VCC connected pin, Ground connected pin and an output pin which is usually connected to pin 2 or 3 so that I2C interrupts can be used to accurately detect the change in clock signals . Fixation of the encoder can bee seen in the following picture :

Fig. 8: Optical encoder

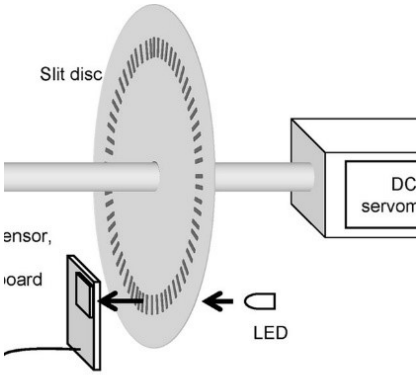


Fig. 9: Optical encoder Principle

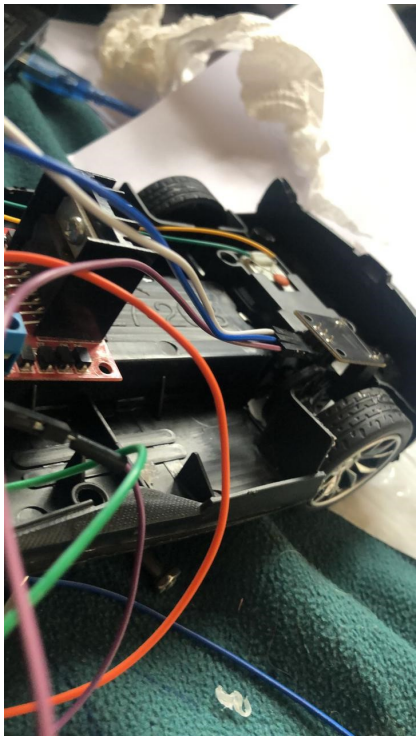


Fig. 10: Optical encoder Principle

C. Gyroscope gy-521

The GY-521 module is a breakout board for the MPU-6050 MEMS (Microelectromechanical systems) that features a 3-axis gyroscope, a 3-axis accelerometer, a digital motion processor (DMP), and a temperature sensor. The digital motion processor can be used to process complex algorithms directly on the board. Usually, the DMP processes algorithms that turn the raw values from the sensors into stable position data. However, we will exploit the GY-521 measurement of yaw angle only to determine the orientation of our car , where it is fixed on the chassis on a smooth square permitted and it has to be fixed with a strong equally distributed glue to prevent any measurements errors . The standard code of the MPU6050 is used to calibrate and filter the readings so that we can extract yaw (theta) .The fixation of the gyroscope is

shown in the picture below . The pinout diagram consists of the following connections :
1. GND pin
2. VCC pin
3. SDA pin connected to A5 pin
4. SLC pin connected to A4 pin

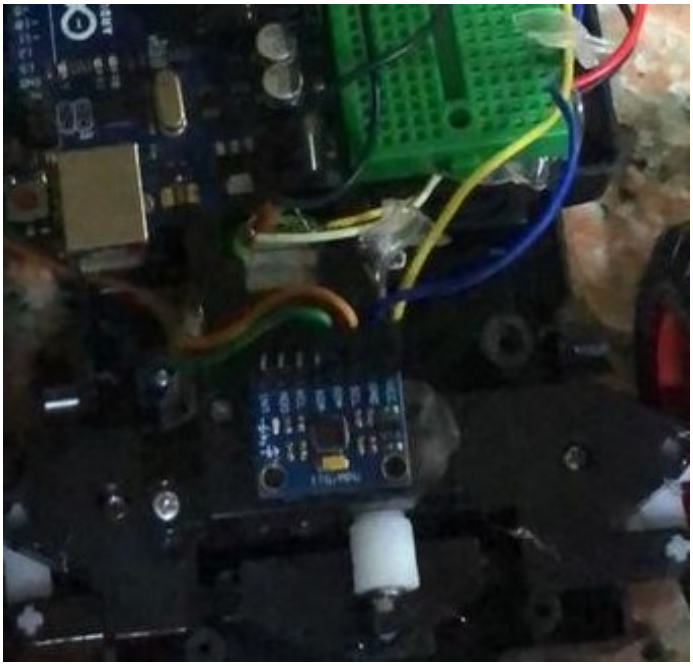


Fig. 11: Gyroscope in the Hardware

D. Integration of the Hardware

After fixing all of the sensors and installing the required voltage sources . H bridge connection to the front steering motor and to the rear motor is soldered perfectly and the USB connection between the raspperybi and Arduino is done , and the sensors data are accurately tested and extracted , the car is now ready for the trying the speed and orientation control phases.

Rasperry Pi is now connected in the loop and ready to start the race.

V. Results and Discussion

The Results have shown full success in the Simulation part after being able to simulate the vehicle model in action as appears here, after being able to navigate and change position, speed, lateral distance on the given map.

For the Hardware part the Arduinos along was the vehicle was able to control the vehicle.

Communication also worked between the multiple platforms.

However the vehicle gearbox was broken at the end of the milestone.

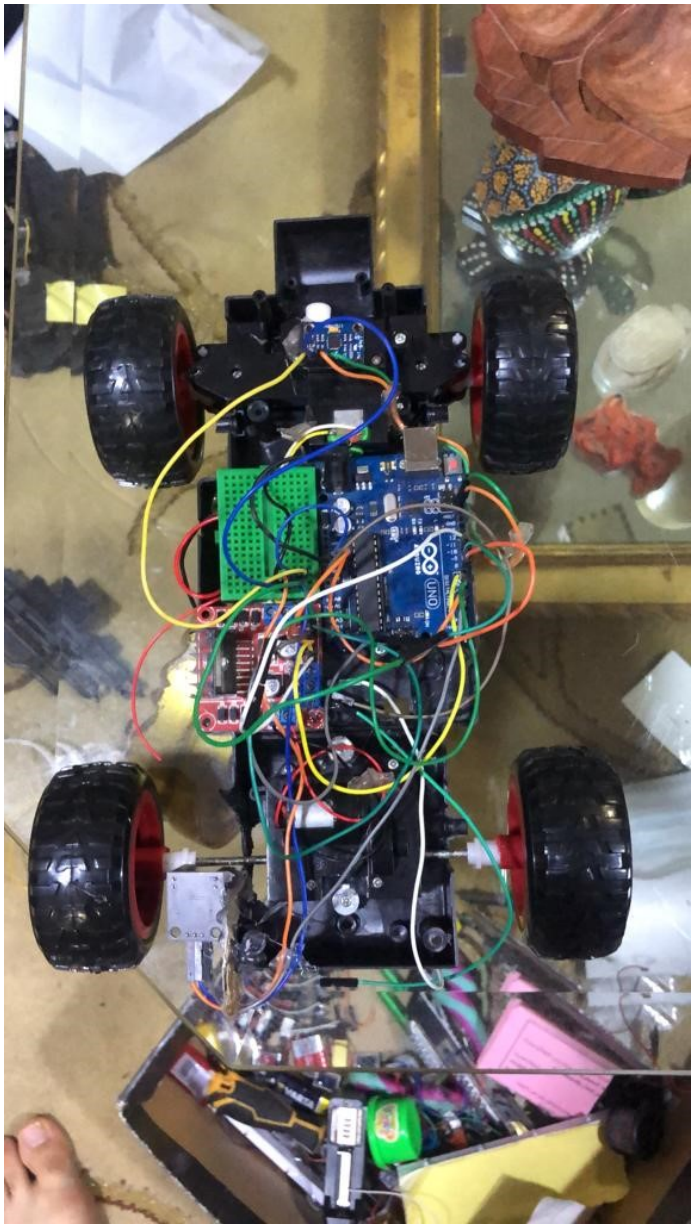
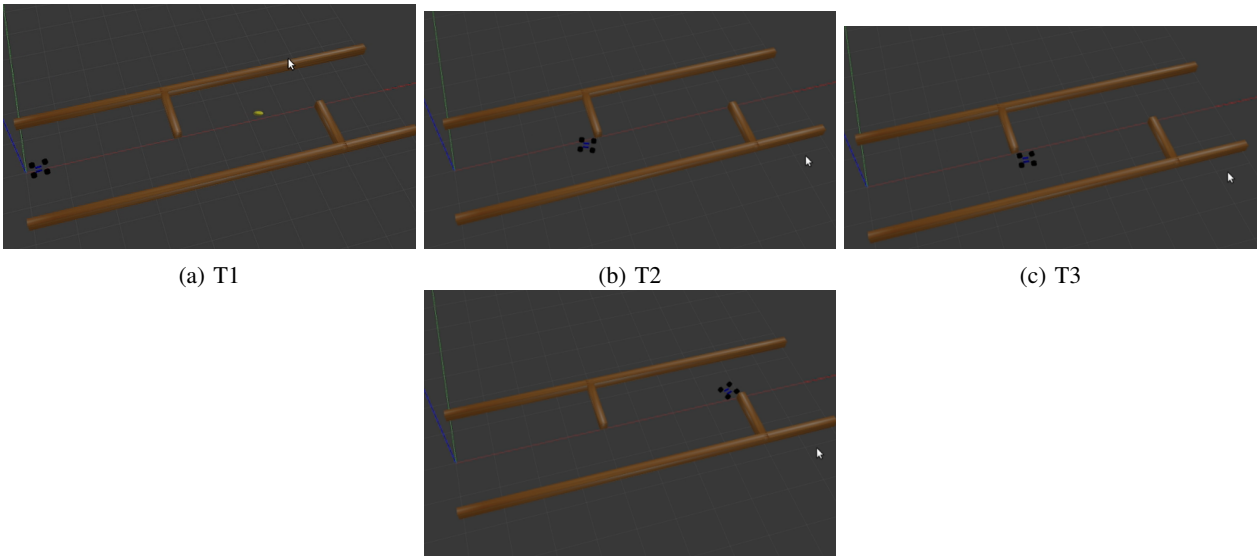


Fig. 12: Full Hardware

VI. Conclusion and Future Recommendations

Ackerman vehicle steering was implemented on an autonomous vehicle in this project. Multiple nodes for speed and lateral control of the vehicle were developed using ROS Melodic and Gazebo simulations, as well as a Kalman Filter to clear the noise from the localization problem. For measurement and control purposes, the hardware was built using a ROS-enabled Raspberry Pi 4 and two Arduinos. The sensory elements included a gyroscope and optical encoders.

In the future recommendation we suggest a better chasis for the vehicle and to implement the communication using only one Arduino.



(a) T1 (b) T2 (c) T3

(d) T4

Fig. 13: Simulation Results

VII. REFERENCES

[1] Feraco, Stefano, et al. ”Combined lane keeping and longitudinal speed control for autonomous driving.” International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Vol. 59216. American Society of Mechanical Engineers, 2019.

Jin, Z. H. A. O., Gaston Lefranc, and Abdelkader El Kamel. ”Lateral control of autonomous vehicles using multi-model and fuzzy approaches.” IFAC
Proceedings Volumes 43.8 (2010): 514-520. [3] H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.

REFERENCES