FIRAT UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
TÜRKİYE

OMNIDIRECTIONAL ROBOT INDOOR LOCALIZATION BY USING VISION SENSORS AND ARTIFICIAL CCs BEACONS WITH COPPELIASIM PROGRAM

Mawj Mohammed Basheer

Master's Thesis

DEPARTMENT OF MECHATRONICS ENGINEERING
Program of Electrical and Electronics Engineering Technologies

APRIL 2022

FIRAT UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
TÜRKİYE

Department of Mechatronics Engineering
Electrical and Electronics Engineering Technologies

Master's Thesis

OMNIDIRECTIONAL ROBOT INDOOR LOCALIZATION BY USING VISION SENSORS AND ARTIFICIAL CCS BEACONS WITH COPPELIASIM PROGRAM

Author
Mawj Mohammed Basheer

Supervisor
Prof.Dr. Mehmet ÇAVAŞ

APRIL 2022
ELAZIG

FIRAT UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
TÜRKİYE

Department of Mechatronics Engineering
Electrical and Electronics Engineering Technologies

Master's Thesis

Title:	Omnidirectional Robot Indoor Localization By Using Vision Sensors And Artificial CCs Beacons With Coppeliasim Program
Author:	Mawj Mohammed Basheer
Submission Date:	24 March 2022
Defense Date:	18 April 2022

THESIS APPROVAL

This thesis, which was prepared according to the thesis writing rules of the Graduate School of Natural and Applied Sciences, Fırat University, was evaluated by the committee members who have signed the following signatures and was unanimously approved after the defense exam made open to the academic audience.

Signature

[redacted]

This thesis was approved by the Administrative Board of the Graduate School on

..... / ..... / 20 .....

Signature

Prof. Dr. Kürşat Esat ALYAMAÇ
Director of the Graduate School

DECLARATION

I hereby declare that I wrote this Master's Thesis titled “Omnidirectional Robot Indoor Localization By Using Vision Sensors And Artificial CCs Beacons With Coppeliasim Program ” in consistent with the thesis writing guide of the Graduate School of Natural and Applied Sciences, Firat University. I also declare that all information in it is correct, that I acted according to scientific ethics in producing and presenting the findings, cited all the references I used, express all institutions or organizations or persons who supported the thesis financially. I have never used the data and information I provide here in order to get a degree in any way.

18 April 2022

Mawj Mohammed Basheer

PREFACE

In this project, an omnidirectional robot that achieves indoor localization by using the algorithm of CCs (color code signatures) as an artificial passive beacon is done by the CoppeliaSim program. The main difficulties of this project were in the design of an artificial passive beacon. In addition, the programming of the whole program in CoppeliaSim.

Mawj Mohammed Basheer
ELAZIG, 2022

TABLE OF CONTENTS

Preface iv
Abstract vi
Özet vii
List of Figures viii
Lıst of Tables x
List of Appendices xi
Symbols and Abbreviations xii

1. INTRODUCTION 1
1.1. Literature reviews: 2

2. BACKGROUND 5
2.1. Robot System 5
2.2. Localization Types 6
2.3. Localization Problems 7

3. OMNI-DIRECTIONAL MOBILE ROBOT SYSTEM 9
3.1. Kinematics Analysis of the Three Omni-Wheels Robot 9
3.2. Control of the Robots 12
3.2.1. Open-Loop Control: 13
3.2.2. Closed-Loop Control 13
3.3. PI Control of the Omni-Wheels Robot: 14

4. MATERIAL AND METHOD 15
4.1. CoppeliaSim Program 15
4.2. Vision Sensor in CoppeliaSim: 16
4.2.1. Composition of Vision Sensor Filters: 18
4.2.2. Depth Map: 20
4.2.3. BLOB Detection: 21
4.3. Lua Programming Language 22
4.4. Kinematics Modeling of Three-Wheels Omnidirectional Robot in CoppeliaSim 23
4.5. Color Signature Detection: 24
4.6. Color Code vs. Color Signature: 26
4.7. Artificial Passive Beacon Design: 27
4.7.1. Color-Coded Beacon Detection Framework 29
4.8. The Position Estimation Problem: 30
4.8.1. Trilateration Modelling: 31

5. RESULTS AND DISCUSSION 32

6. CONCLUSIONS 38
References 39
Curriculum vitae

ABSTRACT

Omnidirectional Robot Indoor Localization By Using Vision Sensors And Artificial CCs Beacons With Coppeliasim Program

Mawj Mohammed Basheer

Master's Thesis

FIRAT UNIVERSITY
Graduate School of Natural and Applied Sciences

Department of Mechatronics Engineering
Electrical and Electronics Engineering Technologies
April 2022, Page: xi + 49

The major goal of this project is to design and implement an indoor localization method for Omni-directional mobile robot based on trilateration technique, Color Code signatures (CCs) as artificial passive beacons, and vision sensor. The robot is guided by the vision sensor, which detects color-coded beacons. The artificial passive beacons are designed as three disks positioned one on top of the others with different sizes and colors (primary colors only, RGB). The three disks share the same center. The designed beacons are placed on the ceiling to be visible from most locations within the room.

The robot's model, controller, and localization method are implemented and evaluated inside the CoppeliaSim environment. The simulation results show that the CCs detecting algorithm enables the robot to discover the designed beacons, achieve an accurate localization and reach the target.

Keywords: : Omnidirectional robot; Localization; CoppeliaSim program; Trilateration.

ÖZET

Coppeliasim Programı İle Görme Sensörleri Ve Yapay CC İşaretçileri Kullanarak Omni-Directional Robot İç Mekan Yerelleştirme

Mawj Mohammed Basheer

Yüksek Lisans Tezi

FIRAT ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

Mekatronik Mühendisliği Anabilim Dalı

Nisan 2022, Sayfa: xi + 49

Bu projenin temel amacı, çok yönlü mobil robot için trilaterasyon tekniğine, yapay pasif işaretler olarak Renk Kodu imzalarına (CC'ler) ve görüntü sensörüne dayalı bir iç mekan lokalizasyon yöntemi tasarlamak ve uygulamaktır. Robot, renk kodlu işaretleri algılayan görüntü sensörü tarafından yönlendirilir. Yapay pasif işaretler, farklı boyut ve renklerde (yalnızca birincil renkler, RGB) biri diğerinin üzerine yerleştirilmiş üç disk olarak tasarlanmıştır. Üç disk aynı merkezi paylaşır. Tasarlanan fenerler, oda içindeki çoğu yerden görülebilecek şekilde tavana yerleştirilmiştir.

Robotun modeli, denetleyicisi ve yerelleştirme yöntemi CoppeliaSim ortamında uygulanır ve değerlendirilir. Simülasyon sonuçları, CC'lerin tespit algoritmasının robotun tasarlanan beacon'ları keşfetmesini, doğru bir lokalizasyon elde etmesini ve hedefe ulaşmasını sağladığını göstermektedir.

Anahtar Kelimeler: Çok yönlü robot; Yerelleştirme; CoppeliaSim programı; Trilaterasyon

LIST OF FIGURES

Page

Figure 2.1. B.Ilon omni-wheel 5
Figure 2.2. The mobile robot system's control scheme 6
Figure 2.3. Schematic mobile robot localization. 7
Figure 3.1. Kinematic modelling diagram of the omni-wheels robot. 9
Figure 3.2. The omni-wheels translation velocity vectors. 10
Figure 3.3. Open-loop control system. 13
Figure 3.4. Close loop control system scheme. 13
Figure 3.5. PI controller system. 14
Figure 4.1. CoppeliaSim program IDE view. 15
Figure 4.2. Industrial robot and Line tracer vehicle equipped with vision sensors. 16
Figure 4.3. Vision sensors types a. Orthogonal -type. b.perspective -type. 17
Figure 4.4. Configuration of the perspective angle. 18
Figure 4.5. The orthographic size with respect to the resolution. 18
Figure 4.6. The vision sensor with three components. 19
Figure 4.7. The image's X-axis is opposite to the vision sensor's X-axis. 20
Figure 4.8. Filter setting. 20
Figure 4.9. BLOB detection. 21
Figure 4.10. The information about the image is returned by the filters. 22
Figure 4.11. Omnidirectional Robot Model. 23
Figure 4.12. Velocity control programming. 24
Figure 4.13. The original image is in the sensor's view. 24
Figure 4.14. Select red color. 25
Figure 4.15. Blob detection for red color. 25
Figure 4.16. Blob detection of three colors (Red-Green-Blue). 26
Figure 4.17. The interference of color signature caught by the vision sensor. 27
Figure 4.18. Beacons are set on the ceiling. 29
Figure 4.19. Blob detection of three colors. 30
Figure 4.20. Trilateration Method Scheme. 30
Figure 5.1. The first scenario of the robot localization. 33
Figure 5.2 . The orientation measurements of scenario 1. 34

Figure 5.3. The Omni-wheels robot X-coordinates of scenario 1. 34
Figure 5.4. The Omni-wheels robot Y-coordinates of scenario 1. 35
Figure 5.5. The second scenario of the robot localization. 35
Figure 5.6. The orientation measurements of scenario 2. 36
Figure 5.7. The Omni-wheels robot X-coordinates of scenario 2. 37
Figure 5.8. The Omni-wheels robot Y-coordinates of scenario 2. 37

LIST OF TABLES

Page

Table 4.1. Designed shape of the beacon. 28

Table 5.1. The result of scenario 1. 33

Table 5.2. The result of scenario 2. 36

LIST OF APPENDICES

Page

Appendix- 1: Derivative of Trilateration method 41

Appendix- 2: CoppeliaSim program and functions 43

SYMBOLS AND ABBREVIATIONS

Symbols

α	: The angel
ω	: Rotational speed
ρ	: Distance
KP	: Proportional Parameter
KI	: Integral Parameter
Vi	: Velocity vector
V_{transi}	: Translation velocity vector

Abbreviations

IDE	: Integrated Development Environment (IDE)
LED	: Light Emitting Diode
GPS	: Global Positioning System
RFID	: Radio-Frequency Identification
EKF	: Extended Kalman Filter
ViPR	: Visual Pattern Recognition
VSL	: Visual Simulation Localization
VSLAM	: Visual Simulation Localization and Mapping
LEA	: Landmarks Exploration Algorithm
CCs	: Color Code Signature
ROS	: Robot Operating System
CCLs	: Color Code Landmarks
PI	: Proportional Integral
PID	: Proportional Integral Derivative
BLOB	: Binary Large Object
API	: Application Protocol Interface
CPU	: Computer Processing Unit
RGB	: Red Green Blue
FOV	: Field Of View

1. INTRODUCTION

Robotics is an academic field that consists of the integration of computer science, electronics, artificial intelligence, mechanics, and control, among others. Mobile robots are considered as a type of robot that has the capability to move around in the workspace fields [1,6]. This item handles mobile robots and how they can move in the real world and achieve their objectives without the need for humans. For the correct operation of mobile robots, there are several technological areas and fields that should be noted and integrated to understand the fundaments such as the locomotion system and kinematics, perception system (sensors), localization system, and navigation system. All these systems must be followed by a control system in order to make the mobile robot achieves its action or task in an intelligible way [26,29].

Mobile robot fundamentals involve the fields of locomotion, perception, cognition, and navigation. Locomotion difficulties can be solved and analyzed by recognizing the mechanism, kinematics, dynamics, and the theory of controls. The responsibilities of cognition are, analyzing the obtained data from the sensors and making the right decisions to fulfill the mobile robot's objectives, which is also responsible for the control of the system. The fields of signal analysis and specialized area like computer vision and sensors fall under perception. Learning about artificial intelligence, planning algorithms, and information theory is recommended for navigation [26].

Localization is considered one of the essential problems that mobile robots face. The best explanation of the localization term is "the ability of the mobile robot to recognize its location within the environment." In other words, the localization term refers to the position and orientation of a mobile robot within a specific location. The localization is divided into two notable kinds [34,35]:

- Relative localization: the robot has to find its location in a local reference frame.
- Global localization: the robot attempts to locate itself in a global reference frame.

Nowadays, indoor localization has become important in different applications such as surveillance, transportation, warehouses, and logistics. These applications contain unclear area that changes infrequently. In such an environment, existing features could be insufficient for mobile robots to generate stable navigation behavior (localization). In order to solve the navigation problem, there are different practical applications depending on passive or active beacons. Beacons make the optimal localization operation easier. In spite of, the artificial beacons are necessary to achieve localization, the number of beacons that are placed in the environment is limited. For the previous reason, a localization algorithm for mobile robots is used, with a specific placement and number of beacons.

Most of the work related to localization is done by using the trilateration method. To localize the robot under certain restrictions, this method needs at least three beacons to be recognized at the

same time, with the distance between them and the robot. In this project, CCs as an artificial passive beacon placed on the ceiling of a corridor frame have been used. These beacons help the Omni-wheels robot to estimate and calculate the position and orientation after applying the trilateration method for indoor localization.

Nowadays, the modern and smart mobile robots are highly expensive, and during the experimental work it can be damaged. Simulators are essential in this field for this reason. Using these simulators to create virtual laboratories has a lot of advantages for robotics research. The CoppeliaSim software is one of the most widely used simulators today, and it deserves special attention. The CoppeliaSim is easy, high scalable and flexible framework for three-dimension simulation in a short time. The CoppeliaSim program has a development environment that is based on script architectural history: each scene object can have an embedded script that runs in a threaded or non-threaded way at the same time. CoppeliaSim simulator includes large number of robot models, simulation experiments, sensors, and actuators which can be used to build a virtual environment and execute it in real time [1]. For the reasons mentioned above in this project, the CoppeliaSim simulation platform has been used.

1.1. Literature reviews:

The mobile robot's localization is considered an important problem for mobile-robot system. For this reason, there are many studies and researches are introduced to solve and design a better system performance. A mobile robot's localization system can be divided into three types. [10], Dead reckoning, Artificial Passive Beacon-based navigation and Map-based navigation. Artificial beacons are often installed at a predefined area of the robot's field. The sensors is used to detect and calculate the orientation and distances between the robot and the beacons, such as cameras or wireless devices [11]. A study on artificial beacons and laser sensors for localization is done by Huosheng Hu and Dongbing Gu [12]. With six identically sized beacons in the workspace field, they utilized a scanner laser with wide range for more than 40m. For successful detection at large distances, the beacons should be made from a single strip and record their information in the robot's memory. These artificial landmarks can provide powerful signals back to the scanner. In addition, they observed that when the mobile robot goes on a non-smooth floor surface, the results changes can be increased due to the laser scanner instability, and this leads to incorrect readings [12]. On the other hand, the location information of the robot is required by the technology of radio frequency identification (RFID). In study of Chunag, L. [13], RFID technique has been utilized to form the tags as an artificial landmark to achieve the localization of robot. These tags have been installed in predefined area within the field of the robot, and the robot's dual-antenna radio frequency reader has successfully communicated with them.

A similar technique was used by Chunag, L. [13] and Gueaieb et al. [14], although they used a different strategy, placing the RFID tags on the ceiling. The main problem with this technology is that exterior items (metal objects or even humans) affect indoor RF propagation; it is extremely sensitive to these items, causing measurements to be changed. Various RFID tags also result in different behaviors. As a result, if this issue is not fixed, the pre-calibrated system may be rendered unusable. As a result of this study, an infrared technology (IR) has been used as an active beacon to lead the robot through an unfamiliar surrounding area [14].

The IR beam-based mobile robot localization system was first presented in 1992. Mobile robot localization applications based on infrared technology are also widely utilized for mobile localization. Before we look at how to use this technology to locate a robot, it's necessary to understand that it's divided into two categories: active beacons and passive beacons. Active beacons are those that produce infrared signals, whilst passive beacons are those that act as reflectors [15].

Beom and Cho [16] utilized a passive beacon with an ultrasonic sensor in 1995. Some elements must be considered reference elements in order to implement localization. When it comes to the choice of the reference element, there are two key elements to consider: First, it must adequately reflect the echo. Second, for the sensor to recognize its location, it must have a feature point. The corner has many reflections which was considered as un-useful reference point (feature point), because of this, it makes the detecting process by the sonar difficult. Furthermore, the sharp edges reflect weak echoes signals, and the position of the wall is difficult to explore since it does not have an identifiable point. Because of the mentioned reasons, [16] has decided to use cylindrical shapes as a passive beacon.

The cylinder's feature point is in the center, which does not change with measurement location. Therefore, locating the robot using a sonar scan requires two cylinders of different diameters. In brief, the location of the cylinder beacon is determined by extracting the distance region from the sensor information. When the sonar is installed on the top of the robot, the sensor will rotate at an increasing angle, emitting ultrasonic pulses and receiving the beacon's echo return. As a result, two encoders are used to measure the signal trip as well as the angles. The heading angle and position of the robot can be measured using information from beacons in the Region of Constant Distance and a mobile robot called LCAR. The primary challenge with this strategy is that if the mobile robot's focal points and both cylinders are collinear, the ultrasonic sensor's measurements will be incorrect, resulting in the estimated distance ($D > 0$) being imaginary. As a result of the sonar's inaccuracy, the error would grow and build over time [16]. Apart from the expanded Kalman filter approach, Kleeman and L. [17,21] utilized ultrasonic as an active beacon using ultrasonic pulses [13].

Active beacons like IR have been employed in the study of Krejsa and Vechet [18]. The IR beacon is set in a predefined location in the surrounding area, and the calculations provide the

orientation between the robot and the beacon. The beacon is made up of 6 light emitting diodes (LED) that emit IR signal, which is detected by the beacon receiver that was fixed on the top of the robot. These LEDs create a 120-degree triggering angle, and as the robot moves, a series of data is generated and sent to the EKF (Extended Kalman filter) estimator, which estimates the robot's location [18]. Although this method performs well in real environments, it has significant drawbacks. Interference is considered as a single source of noise that can affect the calculations, and the fundamental issue with beacons is the limitation of operation distance. This type of beacon is costly because it needs power source, such as a battery to work. In study of Krejsa, V. [18] pointed out, the reason behind utilizing this type of beacons is that being economical. After examining active and passive beacons for localization, it's also appropriate to explore a slightly more sophisticated solution based on optical detecting.

Yamamoto, Pirjanian et al. [19,20] utilized three main technologies and methods to conduct their studies, such as: VSL, ViPR, and vSLAM. Their work has been done with the use of optical beacons (NorthStar) which are considered a low-cost positioning method. NorthStar is an active beacon that has been used as a solution to the localization problem of robots. NorthStar has a unique design that is installed on the ceiling of the room. These beacons can be detected by a vision sensor (that has been set on the top of the robot) depending on the reflection signals from the beacons. In [19], the vSLAM method has been employed. Triangulation has been applied to calculate the pose of the robot in a given environment by using two beacons' locations. The key benefits of this strategy are the low cost of the beacons and the ease of installation with a single projector. North Star localization focuses on the reflected signal of the patterns instead of the transmitter's direct sensing to reduce the line of sight.

Recently, in 2021 [30,31], the researchers used LEA (Landmarks Exploration Algorithm) for indoor localization, which operates in two stages. In the first stage, the artificial CCLs color-coded landmarks (passive cylindrical landmarks) are detected and their locations are stored. At the same time, an extended Kalman filter (EKF) is used to update the robot's state. In the second stage, an proximity sensor is utilized to calculate the distance to the detected CCLs and apply the trilateration method to localize the robot. In addition, [31,33] used the CoppeliaSim environment connected with Matlab for simulation. In this project, CCs (with a circular shape) as an artificial passive beacon placed on the ceiling are used for optimal localization. CoppeliaSim's environment is used for simulation.

2. BACKGROUND

This section will go over the available literature on the robot system and omnidirectional robot. In addition, it contains a review of several ideas for localization and control system.

2.1. Robot System

Nowadays, mobile robots are used in different technical applications and have had a great impact and role in industry fields. Mobile robots can be used for a wide variety of tasks, like transportation and surveillance. One of these robots that play a role in our lives is the omni-wheels robot. The Omni-wheels robot gets its name from the mechanical omni-wheels. In 1973, the first mechanical omni-wheels were introduced by B.Ilon [2]. These omni-wheels enable the robot to move anywhere at any time in the environment [3]. Figure 2.1 shows the first design of omni-wheels. In other words, the omni-wheels give the robot the ability to move in three axes instantly [4].

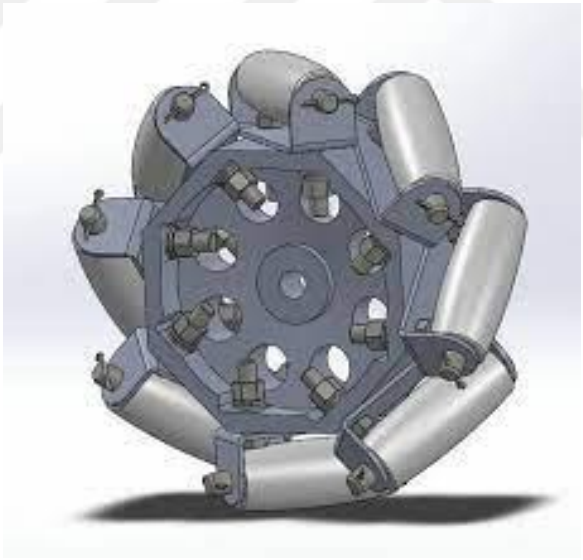


Figure 2.1. B.Ilon omni-wheel

In [5], the omni-wheels robot has flexible and full mobility in the workspace, which gives the robot the capability to move in any direction without re-orienting the robot again. This makes the omni-wheels robot involved and focused on the different robot fields. Most Omni-wheel robots with three omni-wheels have a triangle platform, and these three omni-wheels are considered a good choice for this work due to the fact that each omni-wheel needs only a single motor to be triggered. This design is less costly than the 4 Omni-wheels robots while still providing good mobility.

Even though there are several omni-wheels robot services on the market, the success of the omni-wheels robots depends on their architecture, which combines multiple different knowledge and control components. [2]. Figure 2.2 shows a scheme of the mobile robot system's control. Depending on the application, the functions of the most omnidirectional mobile robot are varied.

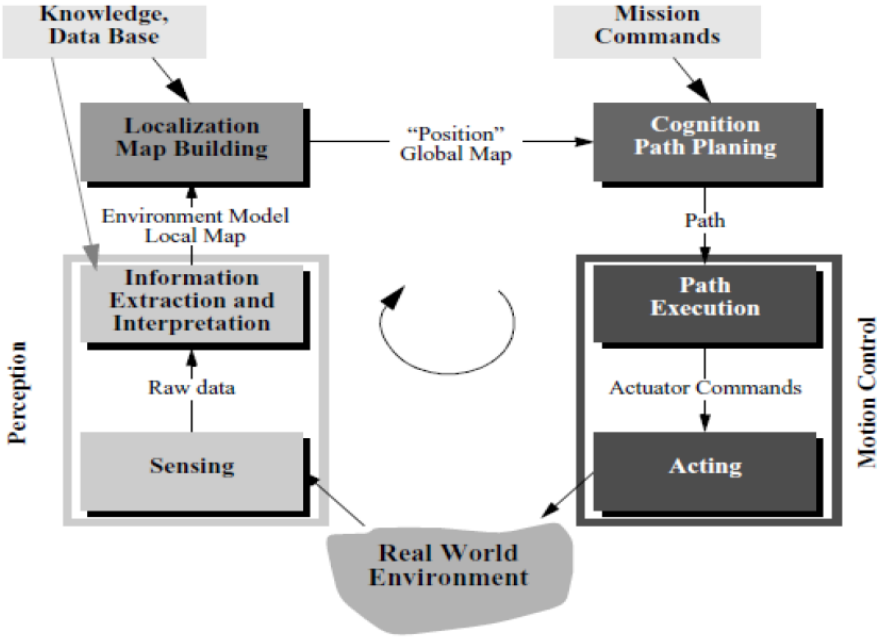


Figure 2.2. The mobile robot system's control scheme

Navigation is the most important function for a mobile robot. There are four points to successful navigation: -

- Perception: gets useful information from the sensor.
- Localization: demands to explore its pose in the surrounding areas.
- Cognition: The method that the mobile-robot uses to reach its target destination.
- Motion control: Represents the model that the mobile-robot uses to control the output of the motor to achieve the desired path.

2.2. Localization Types

Any effective autonomous mobile robot system is dependent on mobile robot localization. Which is essentially divided into two types based on the surroundings:

- Outdoor localization: this type uses GPS because it works well in open fields.
- Indoor localization: GPS is not an appropriate technique for this type of localization because the building outdoors will absorb the signal coming from

satellites and this will make it very weak. For this reason, there are different techniques such as laser range-finding, cell-network localization, and machine vision have been used in this type [7]. Figure 2.3 shows the estimation of robot pose (position and orientation) and implementation of localization. After estimating its current location, the robot should update its (pose) location after each movement step. The previous operation is known as continuous-localization.

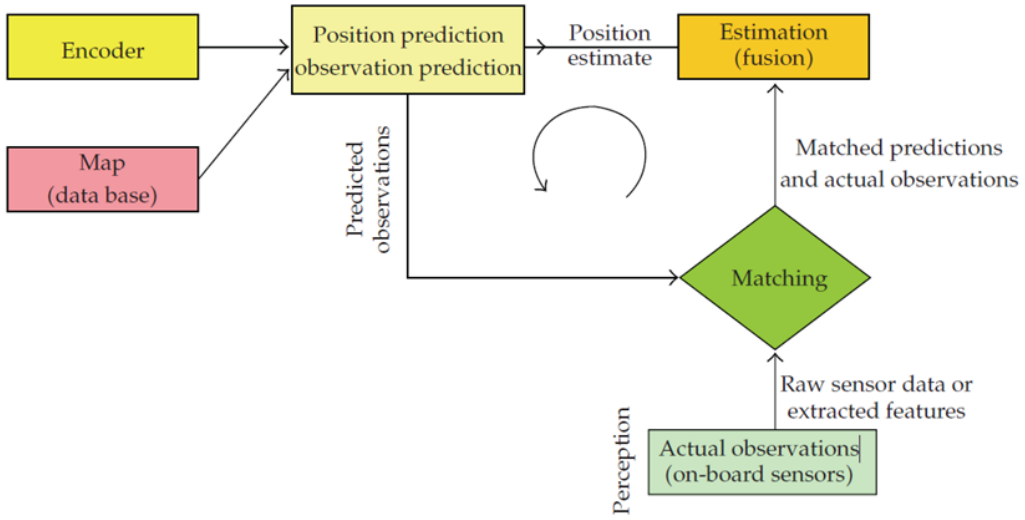


Figure 2.3. Schematic mobile robot localization.

2.3. Localization Problems

Localization problems can be classified into three main parts, which can be divided upon the problem approach that researcher have been adopted [8]:

A. Pose Tracking

The robot needs to keep updating its pose after estimating its initial position by using the readings from the sensors. During the usage of sensors that measure the relative movements, the error will increase over time [9]. For the purpose of estimating the absolute localization of the robot, an external sensor is used for this reason. To determine the absolute position of the robot, the data of the sensors should be matched with the information contained in a map, and this is the most difficult process.

B. Global Localization:

Global localization means that the prediction of the (position and orientation) starts with an insufficiency of information about the initial pose. The factors that affect the process of global

localization that make it complex to estimate are the workspace size and the symmetrical level. The integration of large information sets over time can resolve the symmetrical problems.

C. Map Acquisition:

The processes of location tracking and global localization depend on the variables of the workspace field map. Maps with a pre-defined surrounding building plan are used in some applications. On the other hand, by using the sensor information, a map can be formed in order to serve many applications. Map creation needs the location of the robot to be known. The following points represent the map:

- Topological maps: a map that represents a framework area and is organized in a hierarchical manner.
- Feature maps: Geometric characteristics such as lines and points are used to describe the environment.
- Grid maps: are formed by dividing the environment into small grid-maps. Each single grid-map will represent a small relative area with a general field.
- Appearance based methods: Raw sensor data can be used to define the environment. Raw sensor data provides environmental details that are better than grid mapping or feature mapping.

3. OMNI-DIRECTIONAL MOBILE ROBOT SYSTEM

This section displays the kinematic analysis and mathematical description for three-wheel omnidirectional robot. In addition to the control system types (Open and Close loop) are reviewed.

3.1. Kinematics Analysis of the Three Omni-Wheels Robot

The three Omni-wheels Kinematics modelling is based on the analysis of the Omni-wheels' rotation. Motion control methods can be created depending on the analysis of the Omni-wheels robot's kinematic design [24]. The calculations of the Omni-wheels robot's pose and speed represent the analysis of a kinematics model. Figure 3.1 illustrates the kinematics modelling of the omni-wheels robot and the axis of each (robot and global) frame. To achieve full control of the omni-wheels' motion, the exploration of the kinematics analysis for the omni-wheels is highly required. [25].

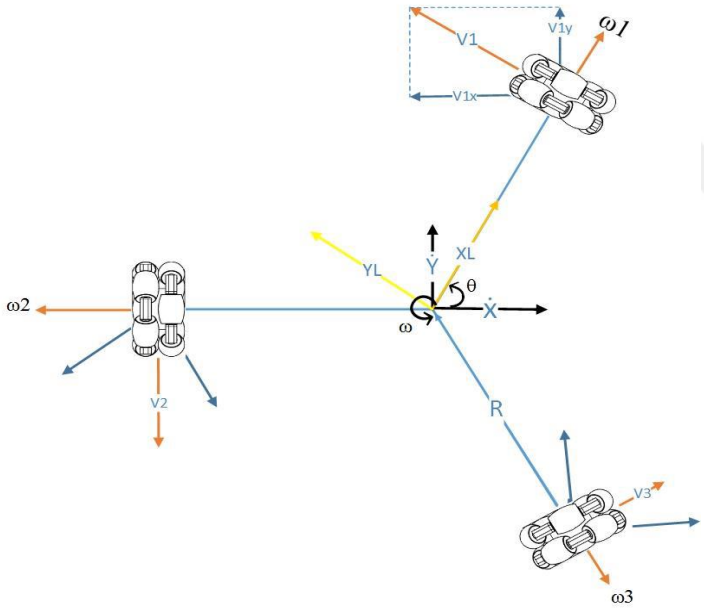


Figure 3.1. Kinematic modelling diagram of the omni-wheels robot.

The global frame can be represented in three dimensions x, y, z which refers to the environment of the robot. The pose of the omni-wheels robot in the environment can be denoted as x, y, θ, and also $\dot{x}, \dot{y}, \dot{\theta}$ represent the environment (global) frame. The robot (local) frame can be represented by x_L, y_L , where the two frames are sharing the same center. The design of the three omni-wheels should be arranged with three angles that can be obtained by determining the difference between each omni-wheel with a 120°, and these angles can be illustrated as, $\alpha 1 = 0°$,

$\alpha 2 = 120°$ and $\alpha 3 = 240°$. The omni-wheels are linked the robot with its surrounding area, for this reason, it considered as the beginning point for the kinematics analysis of the robot. The kinematic analysis of the omni-wheels robot is illustrated in Figure 3.1. The translation velocity of the wheels (vi) can be determined by using the global velocity of the omni-wheels robot in the environment ($\dot{x}, \dot{y}, \dot{\theta}$). The omni-wheels translation velocity (vi) is divided in two primary elements. The first element is rotation of the omni-wheels robot, and second one is the translation of the omni-wheels robot. The translation velocity component is displayed in the equation (3.1)

$$Vi = V_{transi} + V_{rot} \tag{3.1}$$

The translation velocity at the omni-wheel 1 ($Vtrans_i$, 1) is shown in Figure 3.2. If the translation velocity vector at omni-wheel 1 is separated into two vectors ($\dot{x}$,$\dot{y}$), equation (3.2) will be generated.

$$V_{transi} = -\sin\theta\ \dot{x} + \cos\theta\ \dot{y} \tag{3.2}$$

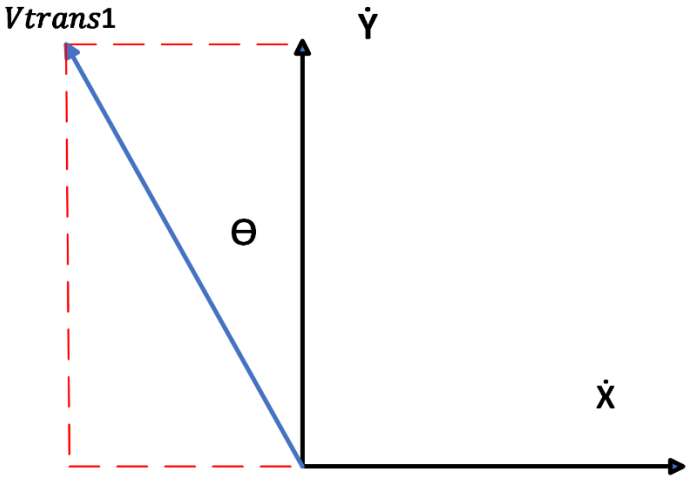


Figure 3.2. The omni-wheels translation velocity vectors.

The rest of the omni-wheels can be calculated by generalizing the equation (3.2). Where $(\theta + \alpha_i)$ represents the offset positioning for each velocity vector v_i, this led to the equation below:

$$V_{transi} = -\sin(\theta + \alpha_i)\ \dot{x} + \cos(\theta + \alpha_i)\ \dot{y} \tag{3.3}$$

When the robot rotates around itself, the omni-wheels translation velocity v_i is required to apply equation (3.4)

$$v_{rot} = R\omega \tag{3.4}$$

R: Represents the distance between the center of the omni-wheels and the center of the robot. Expanding equation (3.1) with the substitute of (3.3) and (3.4), will lead to the following equation:

$$v_i - \sin(\theta + \alpha_i)\,\dot{x} + \cos(\theta + \alpha_i)\,\dot{y} + R\omega \tag{3.5}$$

The angular velocity of the omni-wheels denoted as (ω_i), and the translation velocity are related to each other according to the following equation:

$$v_i = r\,\omega_i \tag{3.6}$$

The term ***r*** is referred to the radius of the omni-wheel. Then, equation (3.7) is obtained by substituting equation (3.6) in (3.5):

$$\omega_i = \frac{1}{r}(-\sin(\theta + \alpha_i)\dot{x} + \cos(\theta + \alpha_i)\,\dot{y} + R\omega) \tag{3.7}$$

This equation can be transformed to:

$$\omega_i = r\,J_{inv}\,\dot{u} \tag{3.8}$$

The inverse Jacobean (J_{inv}) is utilized to provide the relationship between the global velocity vector of the omni-wheels robot ($\dot{u}$), and the angular velocity of the omni-wheels (ω) as illustrated in the following equation:

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} = \frac{1}{r}\begin{bmatrix}-\sin\theta & \cos\theta & R\\ -\sin(\theta+\alpha_2) & \cos(\theta+\alpha_2) & R\\ -\sin(\theta+\alpha_3) & \cos(\theta+\alpha_3) & R\end{bmatrix}\begin{bmatrix}\dot{\mathrm{X}}\\ \dot{\mathrm{Y}}\\ \omega\end{bmatrix} \tag{3.9}$$

In practice, the guiding of the Omni-wheels robot using the environment (global) frame coordination is complex, for this reason, a conversion to the local coordination is desirable. Equation (3.10) display the conversion between the local coordinates and the global coordinates:

$$\begin{bmatrix}\dot{\mathrm{X}}\\ \dot{\mathrm{Y}}\\ \omega\end{bmatrix} = \begin{bmatrix}\cos\theta & 0 & 0\\ 0 & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\dot{\mathrm{X}}_{\mathrm{L}}\\ \dot{\mathrm{Y}}_{\mathrm{L}}\\ \omega\end{bmatrix} \tag{3.10}$$

The integrating between the equations (3.9) and (3.10) leads to the equation below:

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} = \frac{1}{r}\begin{bmatrix}-\sin\theta & \cos\theta & R\\ -\sin(\theta+\alpha_2) & \cos(\theta+\alpha_2) & R\\ -\sin(\theta+\alpha_3) & \cos(\theta+\alpha_3) & R\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & 0\\ 0 & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\dot{\mathrm{X}}_{\mathrm{L}}\\ \dot{\mathrm{Y}}_{\mathrm{L}}\\ \omega\end{bmatrix} \tag{3.11}$$

In order to use equation (3.11) in software environment it is suitable to divide it into three equations as the following:

$$\omega_1 = (-\sin\theta\ \cos\theta\,\dot{X}_L + \cos^2\theta\,\dot{Y}_L + R\omega)/r$$

$$\omega_2 = (-\sin(\theta+\alpha_2)\cos\theta\,\dot{X}_L + \cos^2(\theta+\alpha_2)\cos\theta\,\dot{Y}_L + R\omega)/r \quad (3.12)$$

$$\omega_3 = (-\sin(\theta+\alpha_3)\cos\theta\,\dot{X}_L + \cos^2(\theta+\alpha_3)\cos\theta\,\dot{Y}_L + R\omega)/r$$

In global frame, the forward kinematic velocity should be calculated to reach the full control of the omni-wheels robot and trajectory as follow:

$$\dot{X}_L = r\left(\frac{\omega_3}{\sqrt{3}} - \frac{\omega_2}{\sqrt{3}}\right) \quad (3.13)$$

$$\dot{Y}_L = r\left(-\frac{\omega_3}{3} + \frac{2\times\omega_1}{3} - \frac{\omega_2}{3}\right) \quad (3.14)$$

$$\omega = \frac{r}{3\times1} - \omega_1 - \omega_2 - \omega_3 \quad (3.15)$$

As mentioned above in equation (3.10), the conversion from local frame to the global frame is required to get the global velocity, as shown in the following equations:

$$\dot{x} = \dot{X}_L\cos\theta - \dot{Y}_L\,sin\theta \quad (3.16)$$

$$\omega = \dot{Y}_L\sin\theta - \dot{Y}_L\,cos\theta$$

After the previous mathematical operations, the (x, y, θ) position of the omni-wheels robot can be easily estimated. Also, the trajectory can be plotted by using the integral of the velocities with a specific time period as follow:

$$x = \int_{t_0}^{t} \dot{x}\,.\,dt$$

$$y = \int_{t_0}^{t} \dot{y}\,.\,dt \quad (3.17)$$

$$\theta = \int_{t_0}^{t} \omega\,.\,dt$$

3.2. Control of the Robots

The omnidirectional robot's velocity and position will be controlled by the controller. The control gains and factors that can affect control performance. The control approach is divided into two categories:

3.2.1. Open-Loop Control:

Open-loop control is considered one of the control system types. In this case, the output has no impact on the controlling performance. The system is integrated with an actuator that can be used to run the system without the need for feedback [28]. As a result, when an external disturbance or load condition occurs, it does not respond to the change in the system. The open-loop control system scheme is illustrated in Figure 3.3. This system has various benefits and drawbacks. The benefits include ease of construction and design, low cost, maintenance, and overall stability. On the other hand, the disadvantages include inaccuracy, unreliability, and the inability to automatically correct a sudden change.

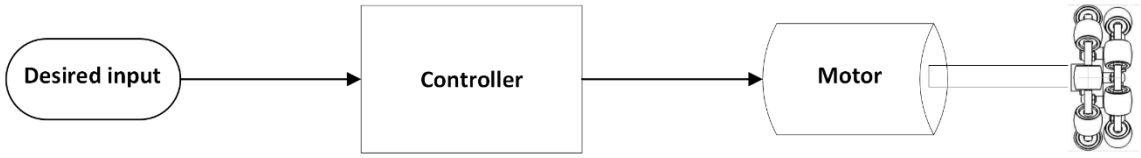


Figure 3.3. Open-loop control system.

3.2.2. Closed-Loop Control

The closed-loop control system is considered another type of control system. This type uses the output to feed the control system to obtain the desired output. The closed-loop control system uses the difference between the actual output and the desired input to achieve the target action [28]. In other words, the closed-loop system is considered an open-loop system with feedback. Figure 3.4 shows the closed-loop control system.

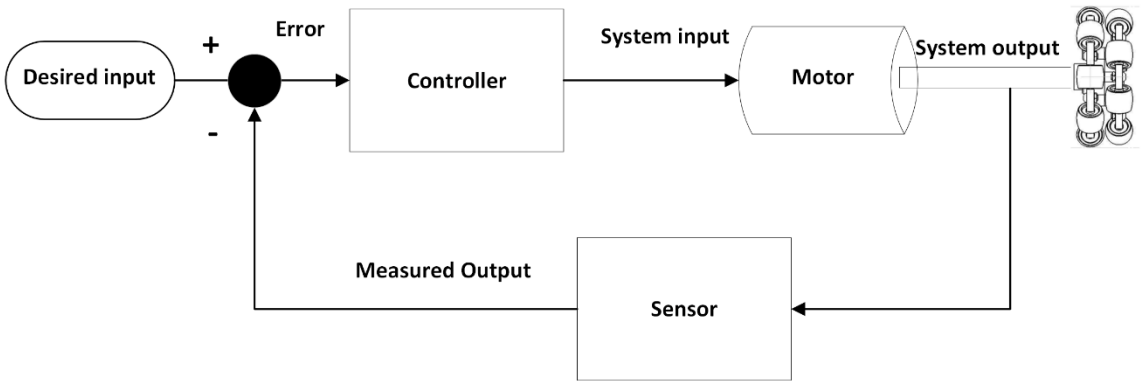


Figure 3.4. Close loop control system scheme.

Closed-loop control system has the following features:

- It is more accurate specifically in the presence of non-linearity.

- The feedback signal is used to correct any immediate change and error automatically.
- Setting the sensitivity of the system to low level, helps the system to work in stable condition.
- Has a low effect by the noise.
- The band-width range of the system is wide.

The mentioned points force the output signal to be close to the desired signal which makes the system to be fully controlled.

3.3. PI Control of the Omni-Wheels Robot:

The error of the controller can be calculated by using the difference between the reference speed and the desired speed. The PI (proportional-integral) controller is a type of PID controller that does not employ the error's derivative (D). The controller output is given by:

$$KP\ e(t) + KI \int e(t)dt \tag{3.18}$$

Where: KP = Proportional gain, KI = Integral gain, and $e(t)$ = the deference between desired-point (DP) and the actual-point (AP). Figure 3.5 shows the PI controller system.

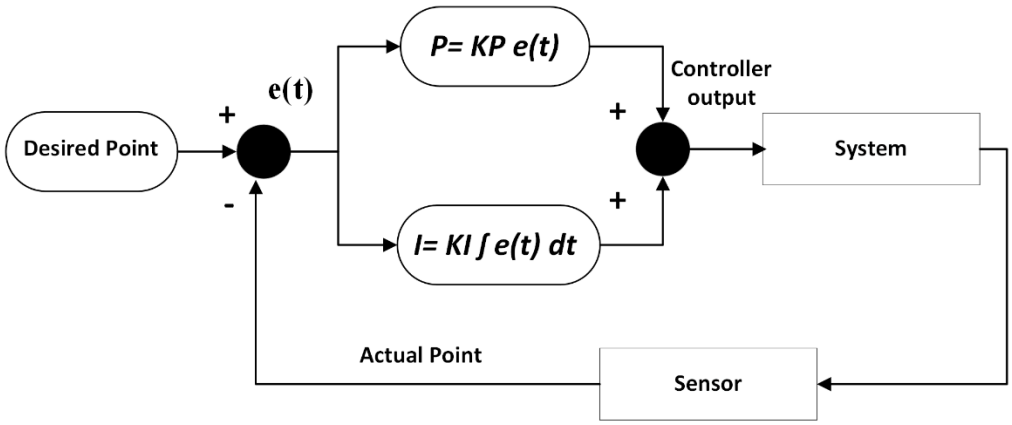


Figure 3.5. PI controller system.

4. MATERIAL AND METHOD

In this chapter, CoppeliaSim environment tools such Omni-directional robot design, Vision sensor, image processing and blob detection filters are explained. In addition, the method that used for localization such as trilateration. Also, the design of the color-coded beacons.

4.1. CoppeliaSim Program

The CoppeliaSim program is considered a new and scalable simulator that is used to simulate three-dimensional programs in a short period of time. In 2010, the first edition of CoppeliaSim was introduced, and it has been updated rapidly. The special features of this simulator make it easy to use in education, especially in robotic fields. The integrated development environment (IDE) in the CoppeliaSim program, which relies on the architecture of distributed control such as an embedded script, a plug-in, a Robot Operating System (ROS) node, a remote API client, or a custom solution, can be used to control each object/model independently. The controllers can be developed using programming languages such as C/C++, Python, Java, Lua, Matlab, Octave, or Urbi. The CoppeliaSim simulator includes a large number of robot models, simulation experiments, sensors, and actuators that can be used to build a virtual environment and execute it in real time [22].The Pioneer 3D-X robot is one of the CoppeliaSim's examples that is used for path following and avoiding the obstacles. The example is shown in Figure 4.1.

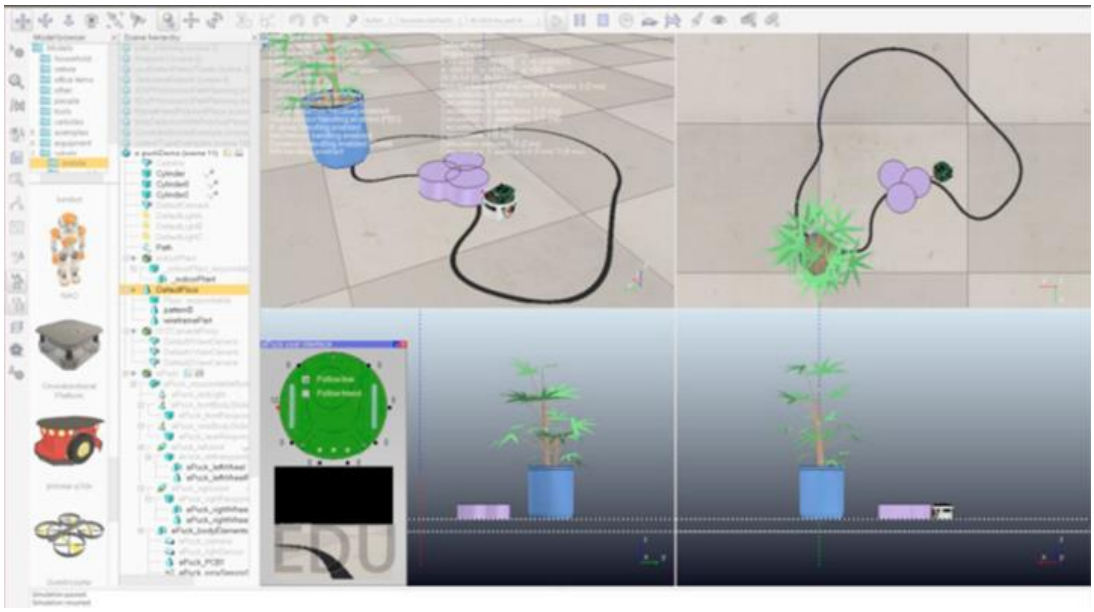


Figure 4.1. CoppeliaSim program IDE view.

Each model in the CoppeliaSim is connected with a script and stored in the software components of the simulator library. These scripts come with infinity loops that can be called by functions to achieve certain tasks inside the simulation program [22]. The programming language used in CoppeliaSim is Lua. It is a scripting language.

4.2. Vision Sensor in CoppeliaSim:

The operation of vision sensors (which are viewable objects) is quite similar to that of camera objects. If defined thresholds are exceeded, they will render the items in their area of vision and trigger detection. When color, light, or structure play a part in the detecting process (e.g., infrared sensors, or, more generally, sensors sensitive to light (cameras, etc.), vision sensors should be utilized over proximity sensors. However, vision sensors may be slower than proximity sensors because their performance is dependent on the internal graphic card, which makes the program run in a complex way [23]. Figure 4.2 shows the vision sensors.

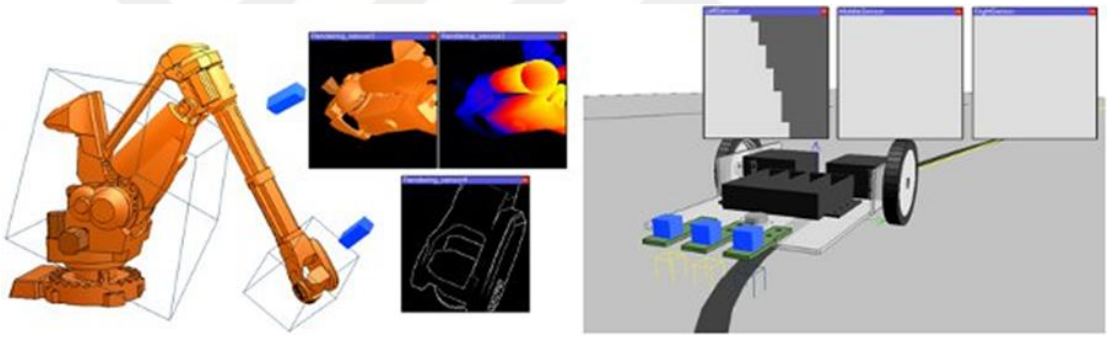


Figure 4.2. Industrial robot and Line tracer vehicle equipped with vision sensors.

There are several differences between the camera (which focuses on the view of the scene) and the vision sensor (which focuses on the processing and visual inspection), despite the fact that the two mentioned devices can display images in the simulator's scene.

A. The resolution of a vision sensor is fixed. There is no such thing as a camera with a specific resolution (i.e., it adjusts automatically to the view size).

B. The API is used to access the image information of the vision sensor, and the filters for image processing are already supported in the vision sensor. On the other hand, the camera uses callback functions to access the image information and it is not supported by image processing filters.

C. A vision sensor, in comparison to a camera, consumes more CPU time and operates at a slower rate.

D. Only renderable objects can be shown by a vision sensor. A camera can show any type of item. (The vision sensor can only detect and analyze items with the renderable attribute set.)

E. Vision sensors can only work when a simulation is operating; this means that a vision sensor's image content is only visible while the simulation is running.

There are two types of vision sensors based on the differences between the projection type and the shapes of view. The first one is the (orthogonal type) and the second one is the (perspective type), as shown in Figure 4.3.

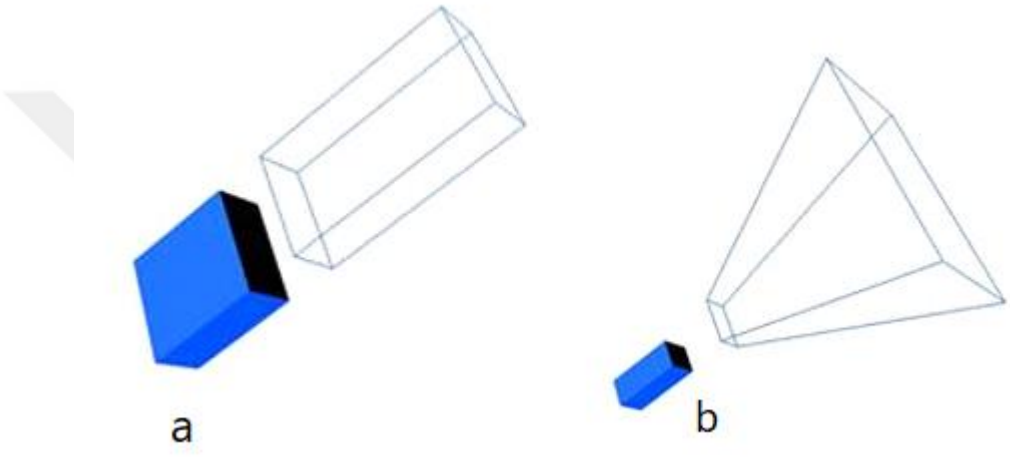


Figure 4.3. Vision sensors types a. Orthogonal -type. b.perspective -type.

A near-clipping plane and a far-clipping plane exist in the vision sensor. The clipping plane can be used to remove certain geometry from a scene and only see or render specific areas of it. Objects closer to the proximal clipping plane than those farther away from the distal clipping plane are not visible. The "Near/Far clipping plane" option in the sensor properties dialog box can be used to adjust the clipping plane's position. The configuration of perspective angle [deg]/Orthographic size can be utilized to configure the sensor's field of view in perspective mode. However, when the vision sensor is working in perspective mode, the configuration of the perspective mode angle is set to the maximum opening angle to detect the volume. As an example, to set the field of view angle to 60° when the *X/Y* resolution is the same as the size of the horizontal and vertical fields of view, as shown in Figure 4.4.

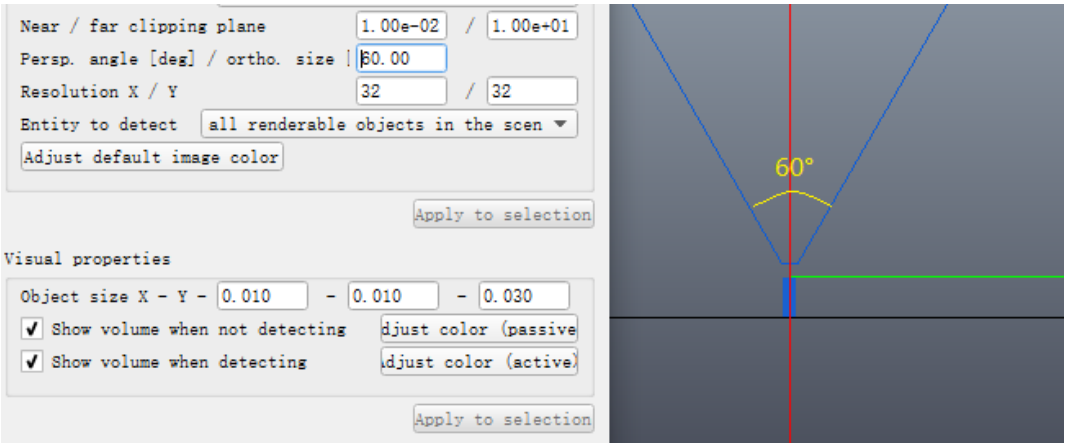


Figure 4.4. Configuration of the perspective angle.

In addition, it can be used to adjust the sensor's field of view size in orthogonal mode. However, the vision sensor is not working in the perspective mode, the configuration of the orthographic size is set to the maximum size to detect the volume (along X or Y). This can be done by setting the orthographic size to 1m, the resolution in the X/Y direction to 64/32, and the X-direction field of view to 1m. the Y direction is 0.5m. As shown in Figure 4.5.

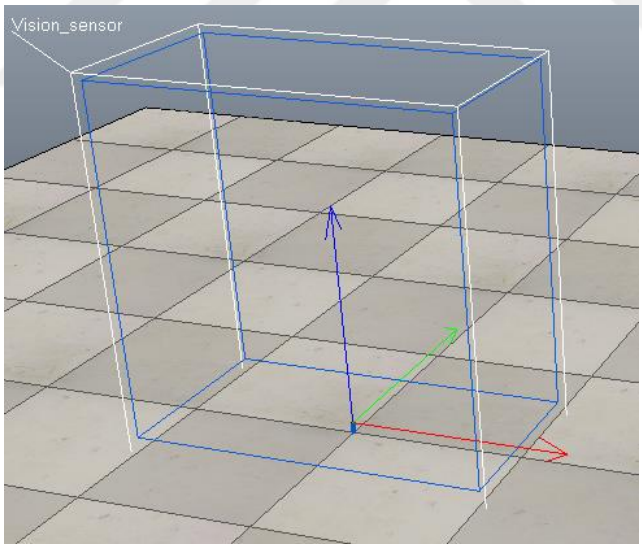


Figure 4.5. The orthographic size with respect to the resolution.

4.2.1. Composition of Vision Sensor Filters:

Image detection and processing are the goals of employing a vision sensor. During the simulation process, the VREP vision sensor can generate two data streams: the depth map and a color image. The entire dataset can be reached by using API calls and then processing each pixel of the image individually. VREP has an internal filter for image processing, which makes using the

built-in filtering and triggering features more convenient and faster. The most basic processing flow of the image consists of three parts as shown in Figure 4.6.

Figure 4.6. The vision sensor with three components.

In the Image processing and triggering dialog box 30 filters can be added for quick image processing, such as:

- Selective color on work image: According to the RGB/HSL value and tolerance, select the specified color in the image, and perform operations such as retention or removal.
- Rotate work image: rotate the image.
- Resize the work image: zoom the image.
- Flip the work image horizontally or vertically: Flip the image horizontally or vertically.
- Edge detection on a working image: edge detection on the image.
- Sharpen the work image: image sharpening.
- Binary work image and trigger: Binary image processing.
- 3×3 / 5×5 filter on work image: Use a 3×3 or 5×5 template to filter the image.

There are several options in the property setting bar of the vision sensor:

- **Ignore RGB info (faster):** If this option is chosen, the sensor can work faster when the color information (RGB) is neglected. This option can be chosen only when the depth information of the sensor is required.
- **Ignore depth info (faster)**: In the case of choosing, the sensor's depth will be neglected, and this can make the sensor work faster. This option can be used when there is no need to use the depth information of the sensor.
- **Packet1 is blank (faster)**: When this option is selected, then CoppeliaSim will not be able to extract the data from the desired image automatically, and therefore the sensor will work in a fast way. This option can be selected when there is no need to use an API to get the stored information in (packet 1).
- **15 auxiliary values (default):** The obtained values from the processing of image pixels contain the maximum, average, and minimum values of intensity, RGB colors, and depth value. However, at a high image processing resolution, the operation of processing will slow down. To reduce the processing time, the values mentioned above can be turned off in the sensor configuration (packet 1). Packet1 will save specific information on the image

(grayscale, RGB, minimum/maximum/average value of depth), which is obtained by traversing all the pixels in the image, so the calculation for images with a large resolution will change slowly.

The *Z* axis of the vision sensor represents the line of sight, where the Y-axis represents the up direction, and the X-axis is perpendicular to the Y/Z axis and points to the left of the sensor. As shown in Figure 4.7. The sensor is placed in front of the robot, it can be seen that the image obtained is the same as that observed by the human eye, and there is no mirror image. The X-axis of the image is opposite to the X-axis of the vision sensor, and the Y-axis is the same.

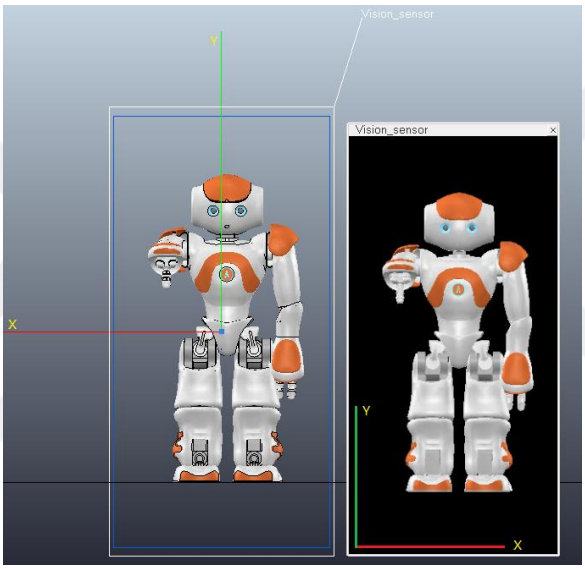


Figure 4.7. The image's X-axis is opposite to the vision sensor's X-axis.

4.2.2. Depth Map:

In order to simply display the depth information, the filter configuration is set as shown in Figure 4.8. The depth map will display on the floating view after starting the simulation.

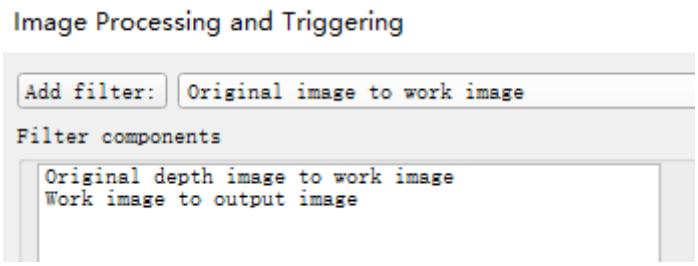


Figure 4.8. Filter setting.

The function (simGetVisionSensorDepthBuffer) is used to get the depth value. Depth buffer one-dimensional table saves the depth data of the specified size on the image. The values in the table represent normalized distances, not actual distances. The sensor's closest value is 0, and the furthest value is 1. As a result, if the clipping plane's position is known, the actual depth can be determined.

4.2.3. BLOB Detection:

The word "blob" is commonly used in computer vision to describe particular parts in an image that have distinct features (e.g., brightness or color) from surrounding regions. Many image analysis methods include the detection of these so-called blobs as a basic component. Blobs can be sometimes object that we need to detect for example medical fields and document texts. Detected regions are usually used as a feedback stage to another stage of detecting the objects, like car movement, monitoring applications using video camera for tracking [32]. In CoppeliaSim, Image processing is divided into two stages:

- Choosing the target color using a selective color filter.
- Detecting and focusing on the target with a Blob Detection filter. As shown in Figure 4.9.

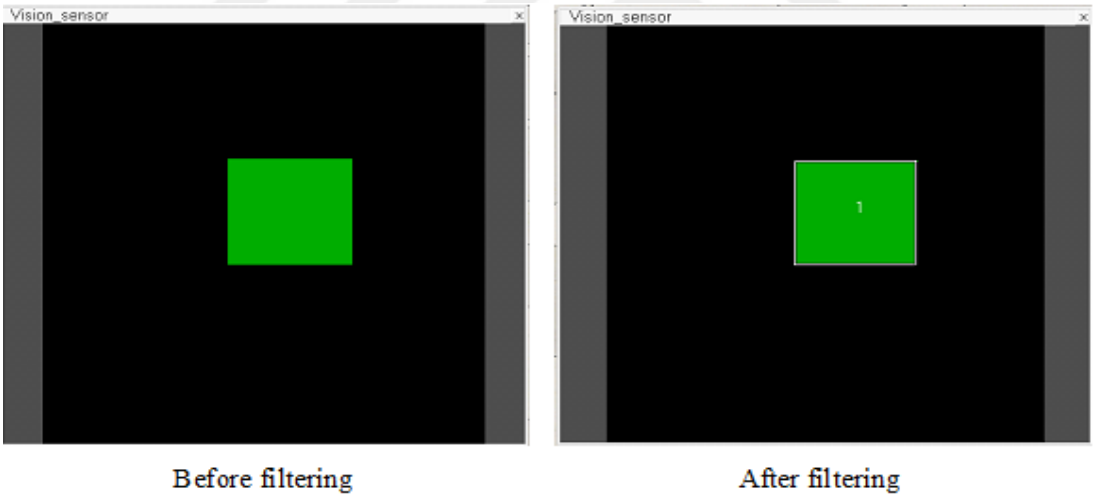


Figure 4.9. BLOB detection.

Some filters can also return some image information, which will be stored in the packets. For example, the blob detection filter can return information about connected areas in the image. Blob detection is to detect the connected domains of the same pixels in an image. Blob analysis is the process of binarizing an image, segmenting the foreground and background, and then detecting connected areas to obtain Blob blocks. As shown in Figure 4.10.

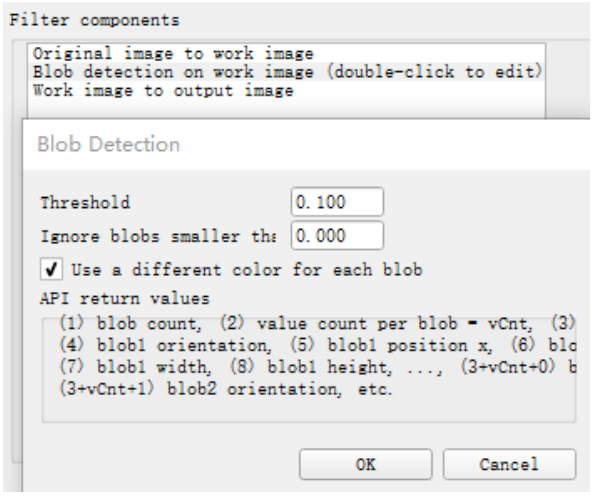


Figure 4.10. The information about the image is returned by the filters.

In CoppeliaSim's scripts, the function (simReadVisionSensor) is used to extract the connected domain information as illustrated below:

{blob count, dataSizePerBlob, blob1 size, blob1 orientation, blob1 position x, blob1 position y, blob1 width, blob1 height, blob2 size, blob2 orientation, blob2 position x, blob2 position y, blob2 width, blob2 height...}.

4.3. Lua Programming Language

Lua is an easy, flexible programming language implemented in C. In 1993, R. Ierusalimschy, and W. Celes produced this language. It was formed from the beginning to be software that could be used in combination with C and other traditional programming languages. There are several advantages to this combination.

It does not attempt to achieve what C already does, but rather focuses on providing what C lacks:

- Good distance from the hardware.
- Dynamical structures.
- There are no overlaps.
- The ease of examining and executing the program's codes.

The Lua language can support a secure framework, automated memory management, and excellent string and other types of data with dynamic sizes. Lua has several unique characteristics that set it apart from other programming languages. Extensible is one of them. Simple, effective, portable. Open and unrestricted [27].

4.4. Kinematics Modeling of Three-Wheels Omnidirectional Robot in CoppeliaSim

In this section. The Omni-Wheels Robot building is done by adding two-cylinder shapes with a diameter of 0.14 m and thickness of 0.005 m as the base of the Omni-Wheels Robot. Three Omni-wheels with a diameter of 0.0352 m and 120 deg between each wheel jointed with the base. The distance between the center of the robot and the center of the wheels has been adjusted with L= 0.093 m. The final model of the Omni-Wheels Robot is shown in Figure 4.11.

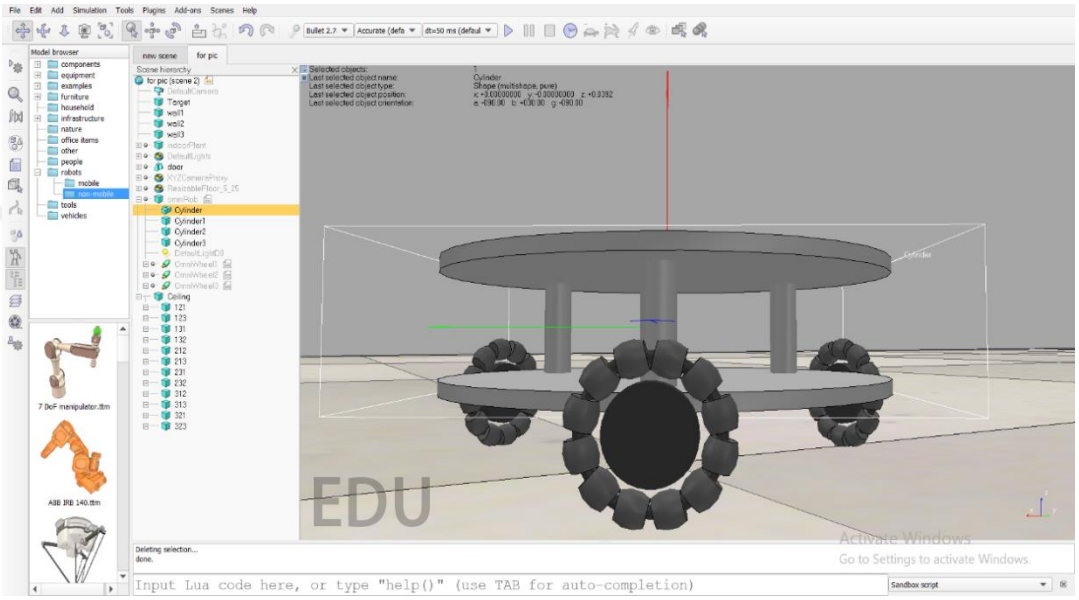


Figure 4.11. Omnidirectional Robot Model.

Since the robot is only moving along the X-direction of the robot's axis, then the velocity in the Y-direction is not necessary can be set to zero. The desired angular velocities of the three Omni-wheels (ω_1, ω_2, ω_3) can be obtained by using the following equations:

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} = \frac{1}{r}\begin{bmatrix}\frac{\sqrt{3}}{2} & -\frac{1}{2} & l\\ 0 & 1 & l\\ -\frac{\sqrt{3}}{2} & -\frac{1}{2} & l\end{bmatrix}\begin{bmatrix}\dot{X}_R\\ \dot{Y}_R\\ \dot{\theta}\end{bmatrix} \tag{4.1}$$

Where r represents the radius of the wheels. l Represents the distance from the center of the robot to the wheels. $\dot{X}_R$, $\dot{Y}_R$, and $\dot{\theta}$ represent the robot's linear velocities along with the robot's X_R, Y_R axes, and angular velocity around its Z_R axis respectively. The wheels' motors are then programmed to track the appropriate angular velocities using a low-level PI controller. The velocity equations are programmed by using Lua in CoppeliaSim's script. Figure 4.12 shows the part of the velocity control program.

```
78          | Xr_dot=-0.1*D_err
79            Yr_dot=0
80            Theta_dot=0
81
82        else
83            Xr_dot=0
84            Yr_dot=0
85            Theta_dot=err*1
86        end
87
88
89  sim.setJointTargetVelocity(omnipads[1], (1/r)*( Xr_dot*math.sqrt(3)/2 -0.5*Yr_dot - L*Theta_dot))
90  sim.setJointTargetVelocity(omnipads[2], (1/r)*(                       Yr_dot - L*Theta_dot))
91  sim.setJointTargetVelocity(omnipads[3], (1/r)*(-Xr_dot*math.sqrt(3)/2 -0.5*Yr_dot - L*Theta_dot))
92
93
```

Figure 4.12. Velocity control programming.

4.5. Color Signature Detection:

In CoppeliaSim, the vision sensor has image processing filters that always search for areas with a given threshold. The RGB color space is employed in this project because it allows for a more detailed segmentation of the required colors than other color spaces. Despite of its undesirable qualities under unbalanced light, it can provide high-performance color detection without any conversion stage [31]. The following experiment in the image processing inside the vision sensor have been used to create the color signature. Where three discs with RGB colored object has been used to show the stages of detection.

- First stage, when the simulation started the vision sensor can display the colored objects. It can be seen on the right-top corner of Figure 4.13.

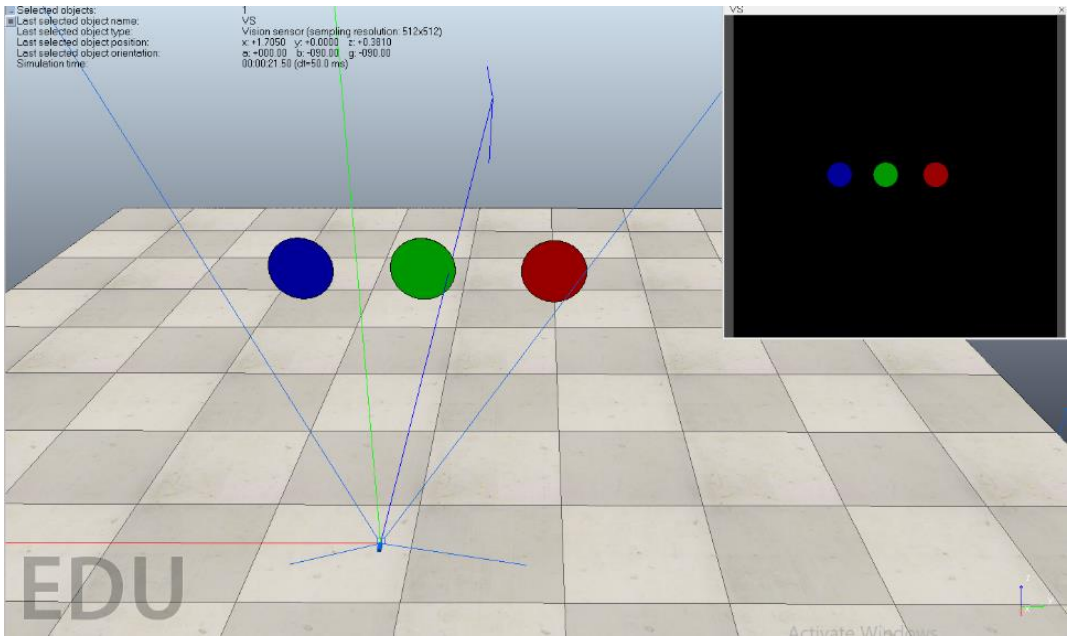


Figure 4.13. The original image is in the sensor's view.

- The second stage, after choosing the target color (Red) using a selective color filter option. As illustrated in Figure 4.14.

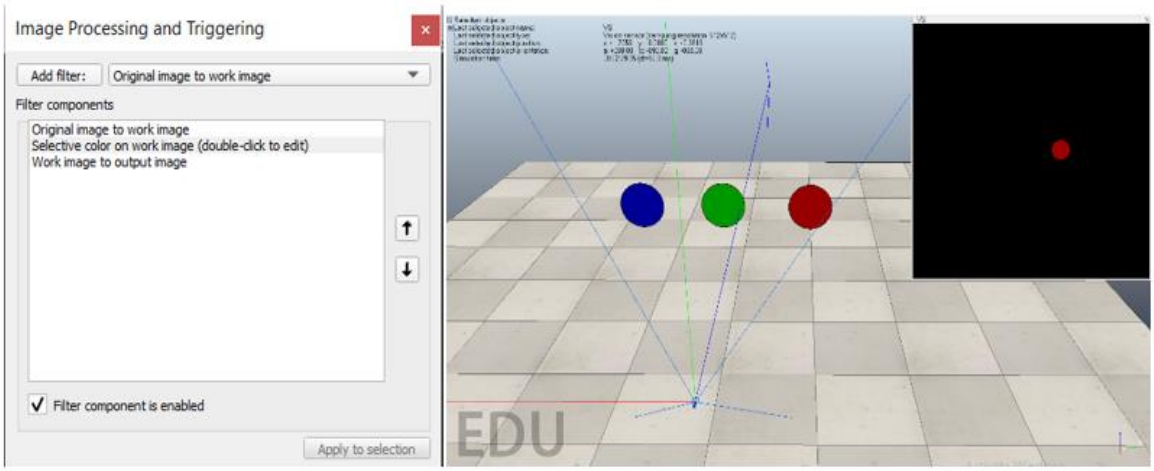


Figure 4.14. Select red color.

- Third stage, Detecting and focusing on the target color (Red) with a Blob Detection filter, as illustrated in Figure 4.15.

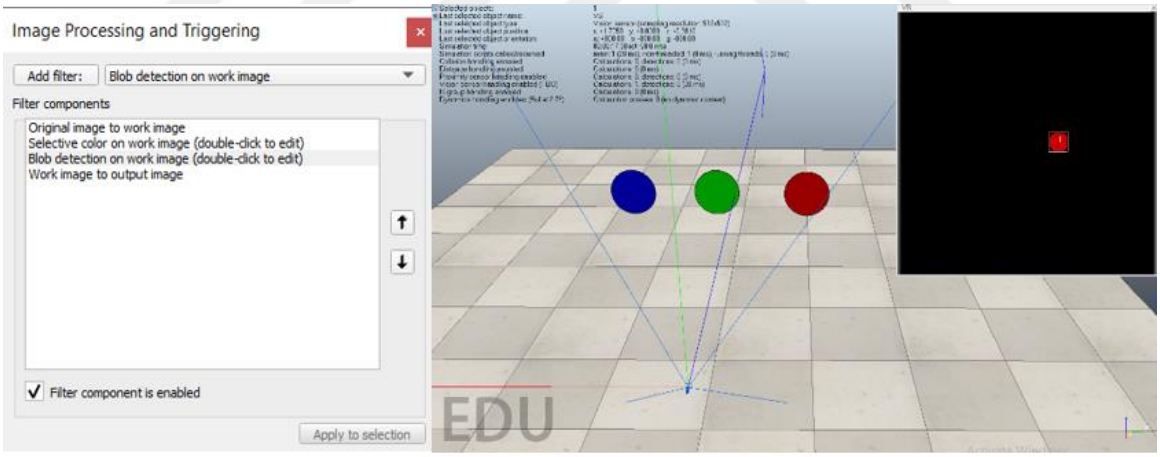


Figure 4.15. Blob detection for red color.

In order to detect other colors (Green and Blue), the three stages above are repeated. Three blob detections for three colors will be generated. As shown in Figure 4.16.

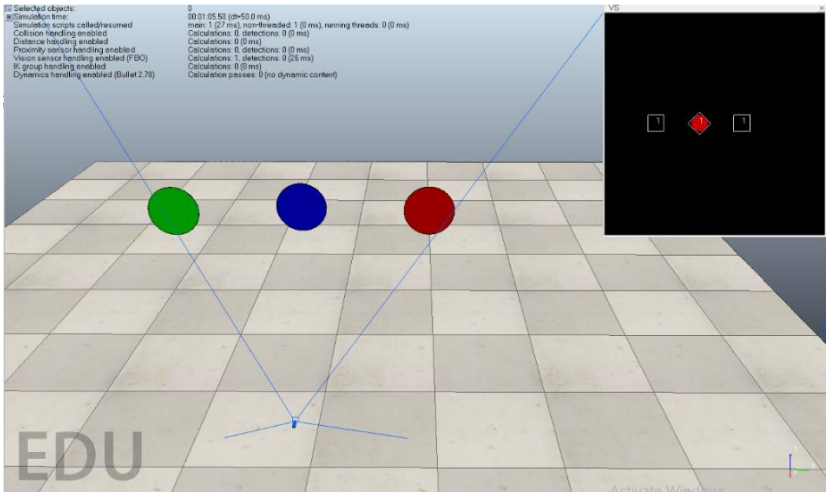


Figure 4.16. Blob detection of three colors (Red-Green-Blue).

In Lua, a CoppeliaSim function (simReadVisionSensor) can be used to get information of detected blobs, such as the number of blobs, their sizes, orientations, and positions. The position of an object inside an image can be obtained with this function, which will be used then in the color code beacon creation**.**

4.6. Color Code vs. Color Signature:

Practically, the color signature is used because it is easy to manipulate and consists of one color. For the mentioned feature, the color signature is used to create the artificial passive beacons. In practical examination of color signatures, the limited numbers of signatures and interference with other objects in the environment have appeared as drawbacks of colors signature. Interference is considered the main issue for a single color. For instance, when a single color is used, such as red, it was expected to get the signature of the selected color (red), but in fact, the sensor caught another object with the same color. This case is illustrated practically by using CoppeliaSim. As shown in Figure 4.17.

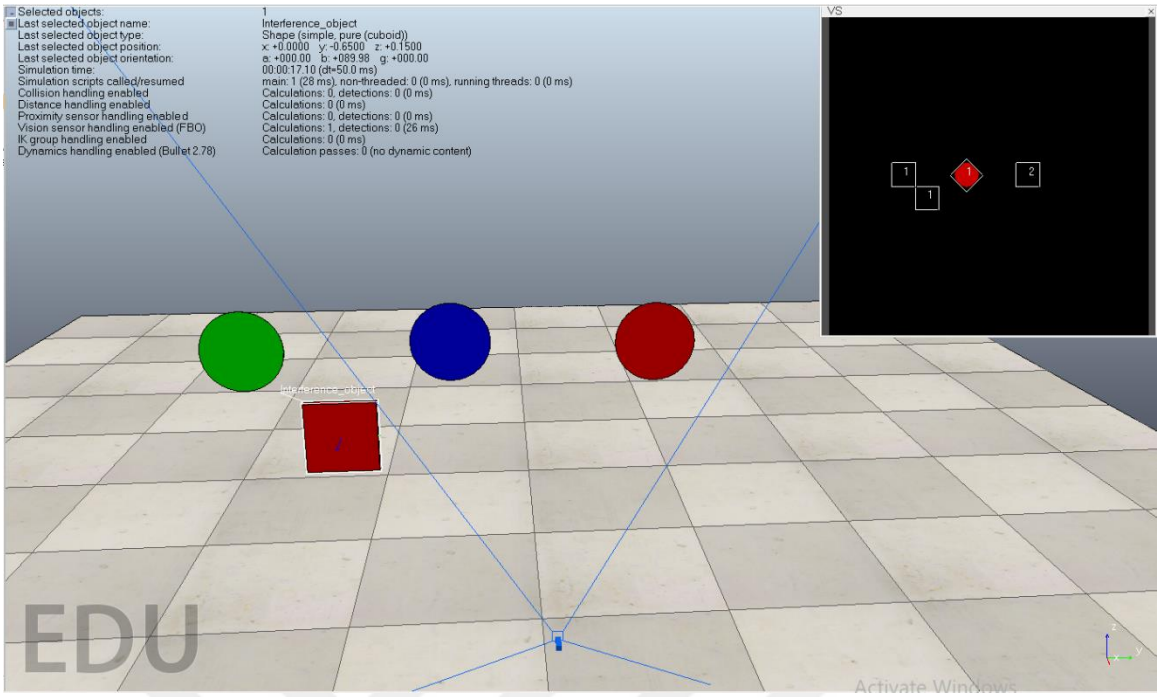


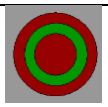
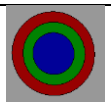
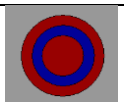
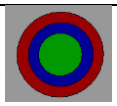
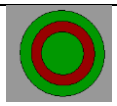
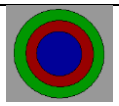
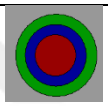
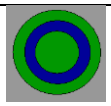
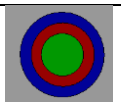
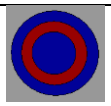
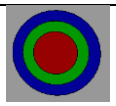
Figure 4.17. The interference of color signature caught by the vision sensor.

In order to overcome the previous problems, color codes can be used as a solution for the color signature. With this color code combining approach, many artificial passive beacons with different color codes and shapes can be generated. These beacons have the property of preventing interference with other colored objects in the workspace. As seen in the practical example in Figure 4.17, that shows the color codes by using the vision sensor.

4.7. Artificial Passive Beacon Design:

Practically, the vision sensor depends on the color's signature to detect the object. In the design of the beacon, the color code signature can be created by adding more than one color signature (CCs) together. The design is based on the combination of three disks with distinct sizes and colors (red, green, and blue only). The disks are placed on top of each other and have the same center. As a result, twelve beacons can be generated. Table 4.1 shows the twelve designed shapes of the beacon and its codes. These artificial passive beacons are placed on the ceiling and used for indoor localization.

Table 4.1. Designed shape of the beacon.

Color Tags	Color Codes	Beacons
Red-Green-Red	121	
Red-Green-Blue	123	
Red-Blue-Red	131	
Red-Blue-Green	132	
Green-Red-Green	212	
Green-Red-Blue	213	
Green-Blue-Red	231	
Green-Blue-Green	232	
Blue-Red-Green	312	
Blue-Red-Blue	313	
Blue-Green-Red	321	
Blue-Green-Blue	323	

The designed beacon is placed on the ceiling with specific arranging that helps the vision sensor to detect at least three beacons at the same time. Figure 4.18 illustrates the arrangement of the designed beacons on the ceiling.

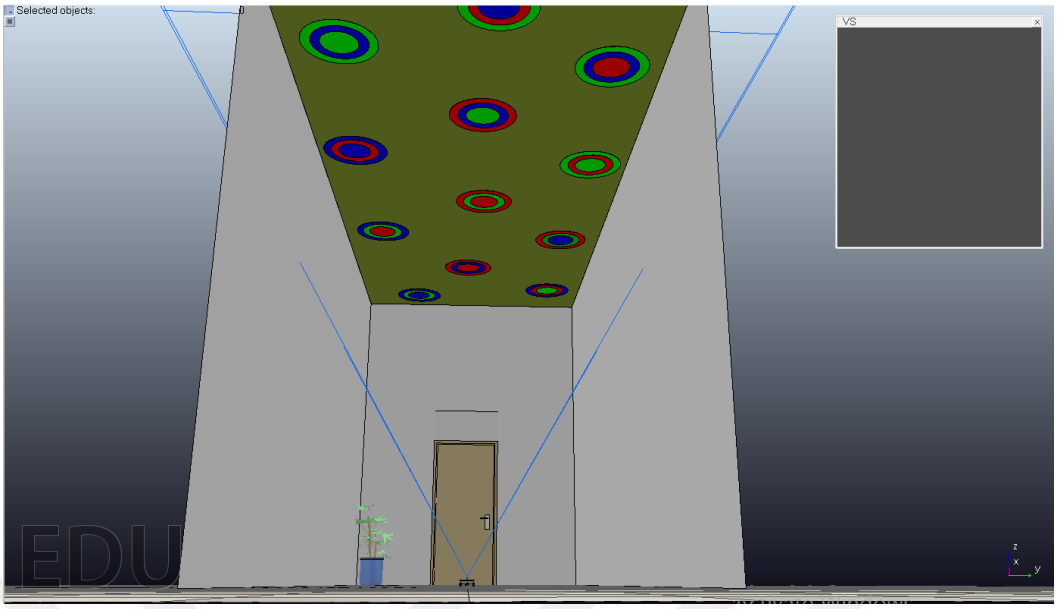


Figure 4.18. Beacons are set on the ceiling.

4.7.1. Color-Coded Beacon Detection Framework

The vision sensor in CoppeliaSim has built-in image processing filters. A combination of these filters is used to detect the designed tag. As shown in Figure 4.19, three filters with appropriate parameters are used to detect only the three primary colors (RGB) in the images and return binary images with one in the pixels with red, green, and blue colors and zero otherwise. After that, three filters are used to segment the images returned from the first three filters. The goal of segmentation is to find the pixels with values of ones that are close to each other, forming a binary large object (BLOB). These filters eliminate the BLOBs with small sizes and also return the coordinates of the center of each blob in the image coordinates in pixels. Each color code signature consists of three blobs of two or three different colors that have the same center coordinates in the image coordinates.

The detection of the CCs beacon is as follows: First, the distance between a blob center of a certain color and every other blob center of the same or different color is calculated using the distance formula between two points. If this distance is less than a certain threshold then the three blobs belong to the same beacon. The process is repeated until all tags are detected and recorded. Second, a color code is assigned to each beacon based on the sizes of the blobs with the assumption that red = 1, green = 2, and blue = 3, see Table 4.1.

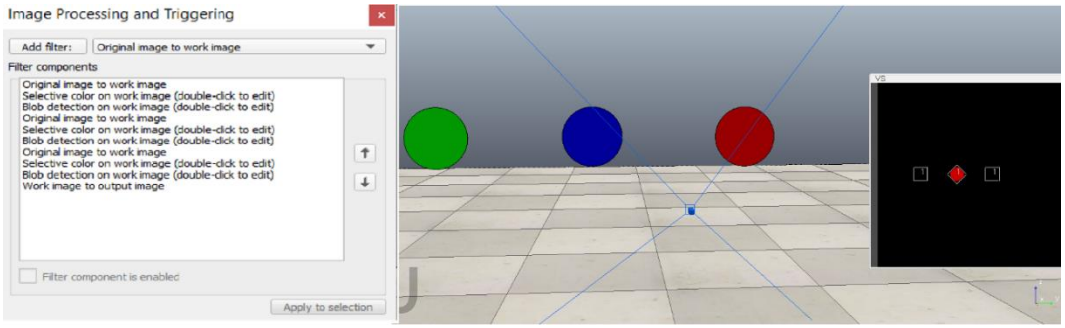


Figure 4.19. Blob detection of three colors.

4.8. The Position Estimation Problem:

Localization means that the omni-wheels robot needs to find its location (position and orientation) in the surrounding area. The localization of the omni-wheels robot can be calculated by applying the trilateration method [7]. As shown in Figure 4.20.

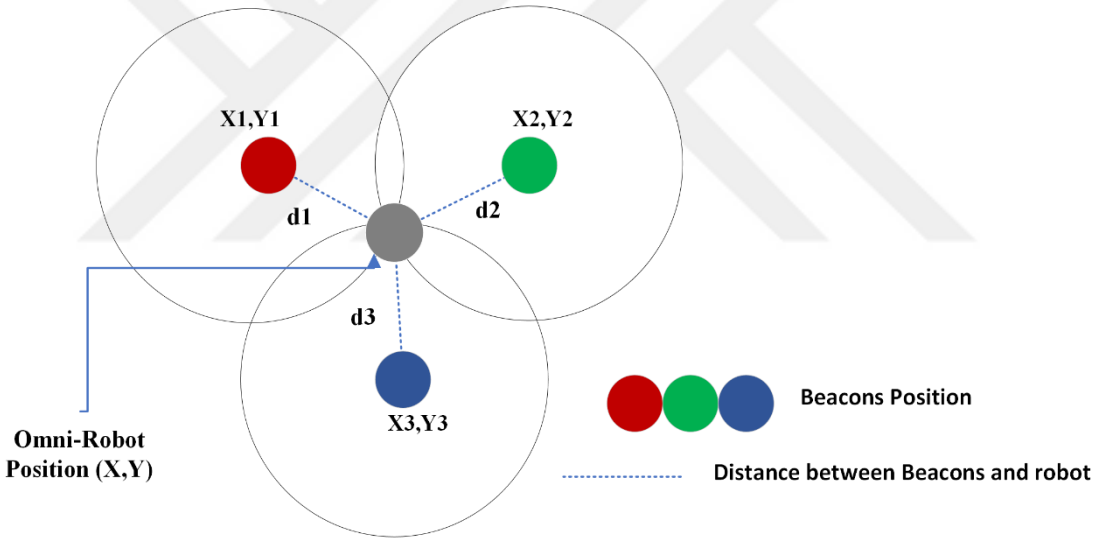


Figure 4.20. Trilateration Method Scheme.

Based on the general equation of the circle, as shown in equation (4.2):

$$d^2 = X^2 + Y^2 \tag{4.2}$$

Where d is the radius of the circle. For a circle centered at a point (X_1, Y_1), (X_2 , Y_2) and (X_3,Y_3). By substituting these centers in the equation (4.2), this step will lead to the equations (4.3), (4.4) and (4.5):

$$d_1^2 = (X - X_1)^2 + (Y - Y_1)^2 \tag{4.3}$$

$$d_2^2 = (X - X_2)^2 + (Y - Y_2)^2 \tag{4.4}$$

$$d_3^2 = (X - X_3)^2 + (Y - Y_3)^2 \quad (4.5)$$

The distances (d_1), (d_2), and (d_3) are known from the depth information of the camera. After the detection of the beacons, their coordinates (X_1, Y_1), (X_2, Y_2), and (X_3, Y_3) of each artificial passive beacon with respect to the global frame are known, which are set by the creator. Therefore, the two unknown variables (X, Y) which represent the point of intersection of the three circles and the robot's current position can be determined by solving the equations (4.3), (4.4), and (4.5).

4.8.1. Trilateration Modelling:

The previous equations of trilateration are hard to understand and to program. Therefore, a mathematical conversion and derivation are required to simplify the equations (4.3), (4.4), and (4.5):

$$\mathrm{X} = \frac{\mathrm{CE} - \mathrm{FB}}{\mathrm{EA} - \mathrm{BD}} \quad (4.6)$$

$$\mathrm{Y} = \frac{\mathrm{CD} - \mathrm{AF}}{\mathrm{BD} - \mathrm{AE}} \quad (4.7)$$

More information on the derivative of trilateration equations is explained on Appendix 1. These equations can now be simply implemented in CoppeliaSim, and the robot can rely on the artificial CC Beacons coordinates received by the vision sensor to determine the robot's exact position.

5. RESULTS AND DISCUSSION

In this project, the localization of the Omni-directional robot has been implemented using the simulation environment of CoppeliaSim which is controlled via scripts written in Lua language. Twelve circular objects with RGB color are used as beacons with a predefined position relative to the global frame. The scene in CoppeliaSim has a corridor with dimensions (3x8x4) in meters. The Omni-wheels robot designed in section (4.4) is used. The vision sensor is located on the top of the robot with a 60˚ FOV to detect at least three beacons at the same time. Two scenarios are considered to show the impact of the suggested localization approach. The process and programing of the color-coded beacon exploration algorithm can be seen in Appendix 2.

The first scenario is shown in Figure 5.1. The robot is located at point **A,** and the target is located at pose ($x_t = -3$ m, $y_t = 1\ m$,$\theta_T = -11.6$˚). In the first stage, the robot at point **A** starts rotating using the following control law:

$$\dot{\theta} = k_{p\theta}(\theta - \theta_d) \tag{4.8}$$

Where θ is the current robot orientation,$\theta_d = atan2(y_t - y_r, x_t - x_r)$, (y_r, x_r) and (y_t, x_t) is the Cartesian coordinates for the robot and target positions respectively and the controller parameter $k_{p\theta} > 0$. In the second stage, trilateration equations (4.6) and (4.7) are applied to obtain the robot's position relative to the global frame. The robot is then moved with a velocity $\dot{X}_R$that is calculated according to the equation:

$$\dot{X}_R = K_P\, \rho\, \cos\alpha \tag{4.9}$$

Where $\dot{X}_R$ is the desired speed of the robot (with the assumption of $\dot{Y}_R = 0$), and the controller parameter $K_P > 0$, $\rho = \sqrt{(y_t - y_r)^2 + (x_t - x_r)^2}$ and $\alpha = 0$ due to the previous rotation by θ_d. Finally, the robot is rotated to achieve the desired orientation θ_T using the control law (4.8) with $\theta_T = \theta_d$ the results are shown in Table 5.1.

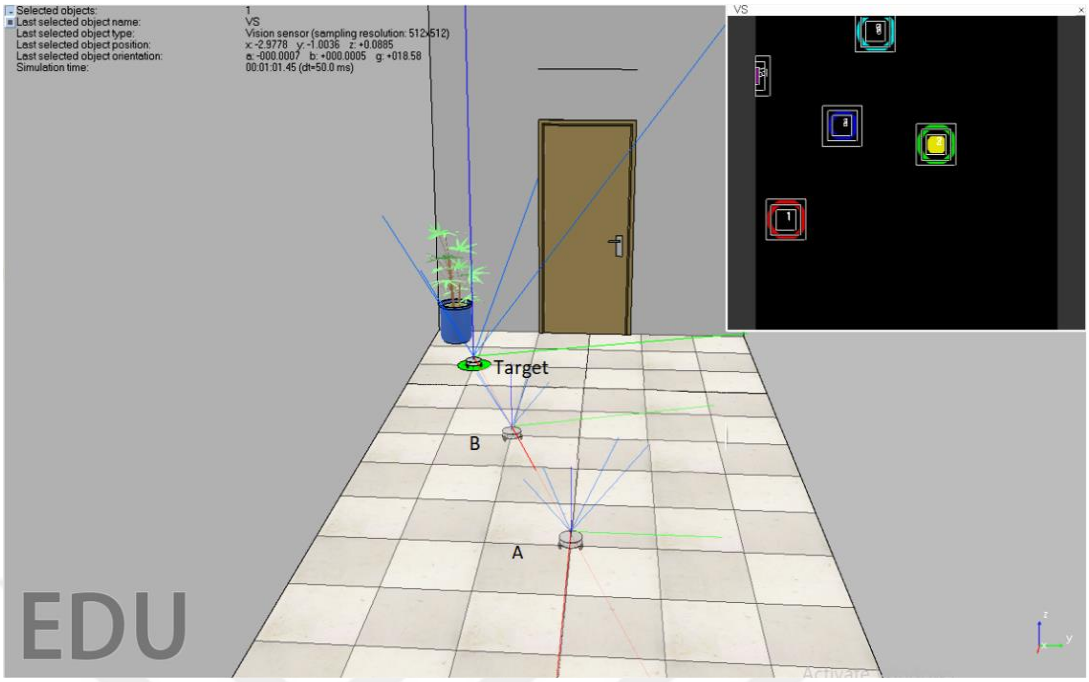


Figure 5.1. The first scenario of the robot localization.

Table 5.1. The result of scenario 1.

	X_{Actual}(m) (Robot)	X_{Tri}(m) (Robot	Y_{Actual}(m) (Robot)	Y_{Tri}(m) (Robot)	Distance error (m)= Target position – Robot position
A	0	0.08	0	0.02	3.16
B	1.227	1.209	0.413	0.404	1.88
Target	3	2.95	1	0.99	0.05

Figure 5.2 shows that The Omni-Wheels robot starts rotating from its initial orientation until reaching θ_d with an execution time of 4 seconds.

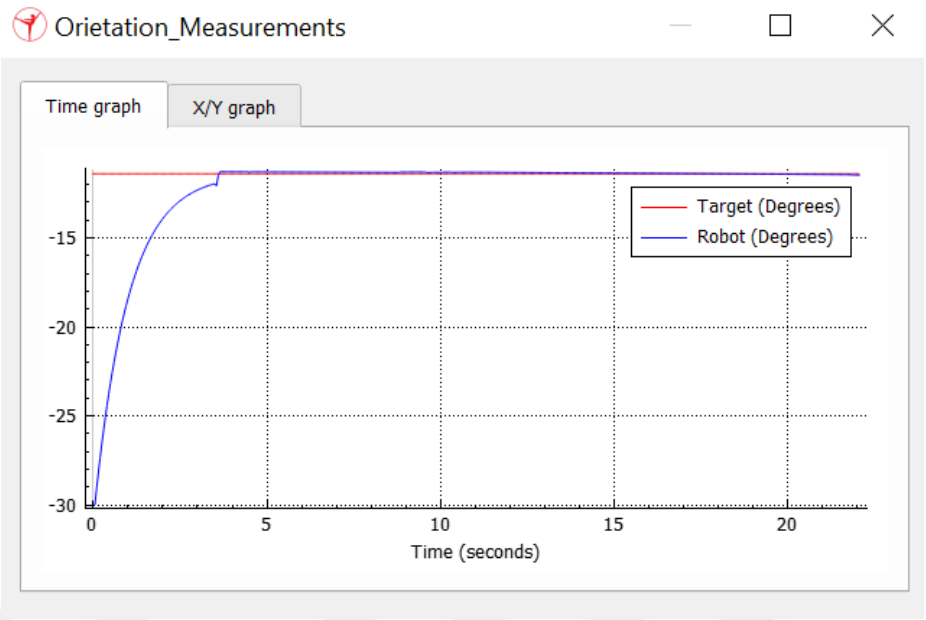


Figure 5.2 . The orientation measurements of scenario 1.

Additionally, the (X, Y) coordinates of the robot from the initial position until reaching the target are shown in Figure 5.3. and Figure 5.4. The graphs show that the robot coordinates do not change for the first 4 sec due to the rotation of the robot; after that, it can be seen that the robot (X, Y) coordinates change exponentially until reaching the target's coordinates.

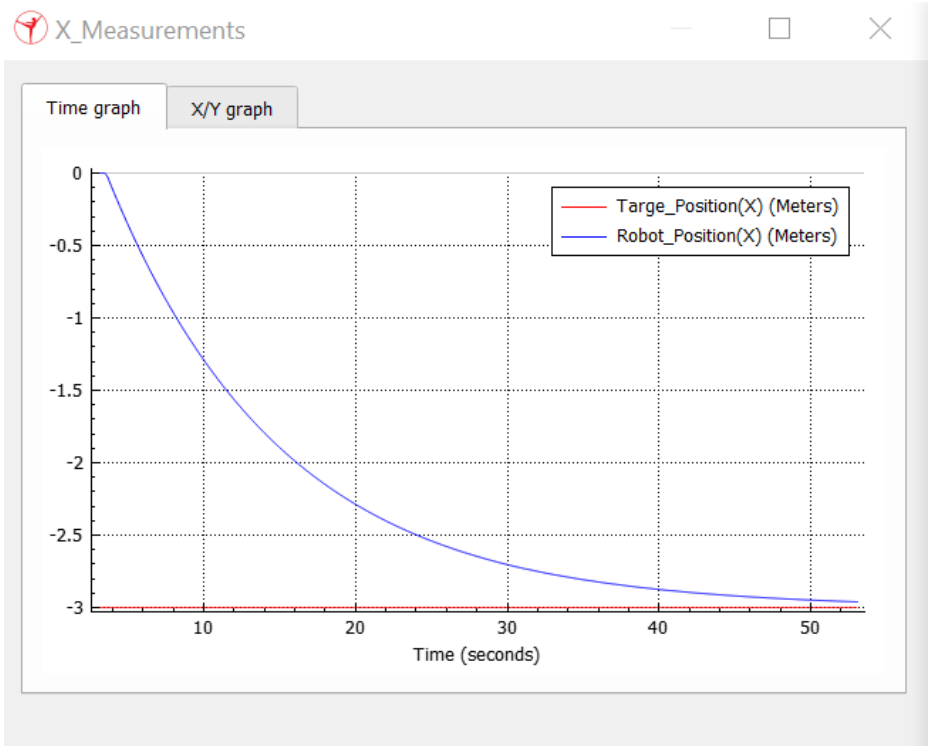


Figure 5.3. The Omni-wheels robot X-coordinates of scenario 1.

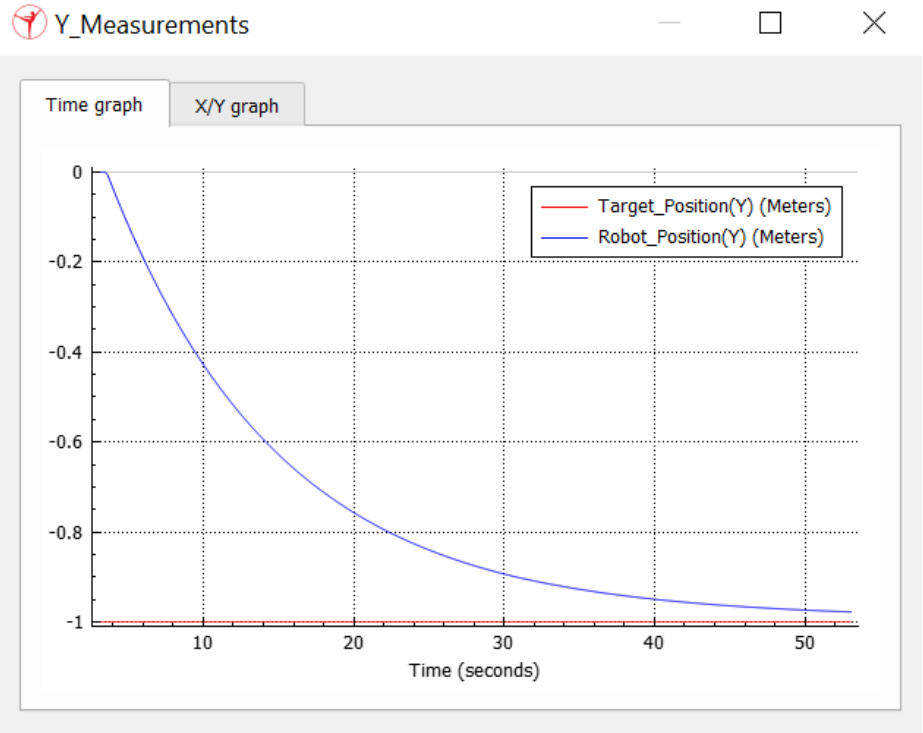


Figure 5.4. The Omni-wheels robot Y-coordinates of scenario 1.

In the second scenario, as shown in Figure 5.5, the target is located at the pose ($x_t = 2$ m, $y_t = 1\ m$,$\theta_T = 176^{\circ}$). The robot starts rotating at point **A** and then the steps of the first scenario are repeated. The robot reaches the target as demonstrated in Table 5.2.

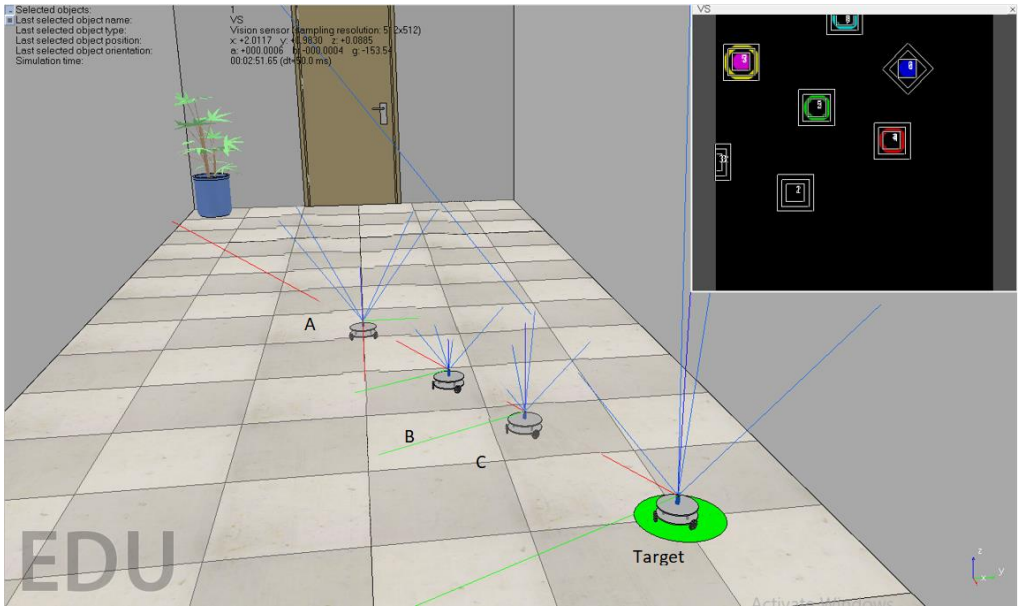


Figure 5.5. The second scenario of the robot localization.

Table 5.2. The result of scenario 2.

	X_{Actua}(m) (Robot)	X_{Tri}(m) (Robot)	Y_{Actual}(m) (Robot)	Y_{Tri}(m) (Robot)	Distance error (m)= Target position – Robot position
A	0	0.039	0	0.019	2.192
B	0.693	0.673	0.339	0.329	1.487
C	1.295	1.288	0.637	0.633	0.801
Targe	2	1.991	1	0.978	0.023

Figure 5.6 shows that The Omni-Wheels robot starts rotating from its initial orientation until reaching θ_d with an execution time of 5 seconds.

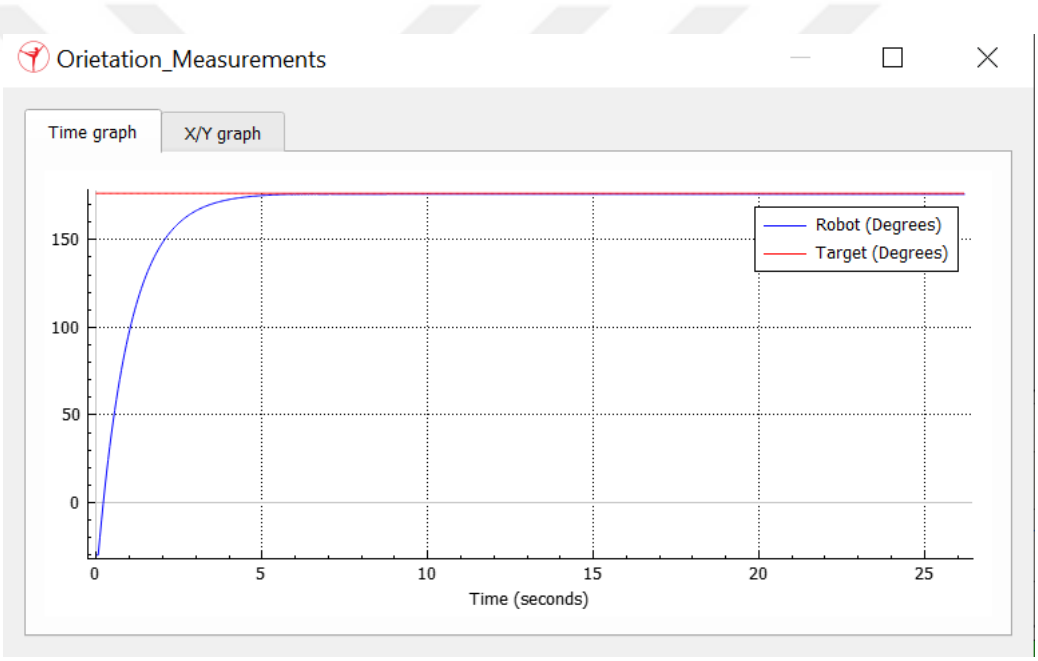


Figure 5.6. The orientation measurements of scenario 2.

Additionally, the (X, Y) coordinates of the robot from the initial position until reaching the target are shown in Figure 5.7. and Figure 5.8. The graphs shows that the robot coordinates do not change for the first 5 seconds due to the rotation of the robot; after this time, it can be seen that the robot (X, Y) coordinates change exponentially until reaching the target's coordinates.

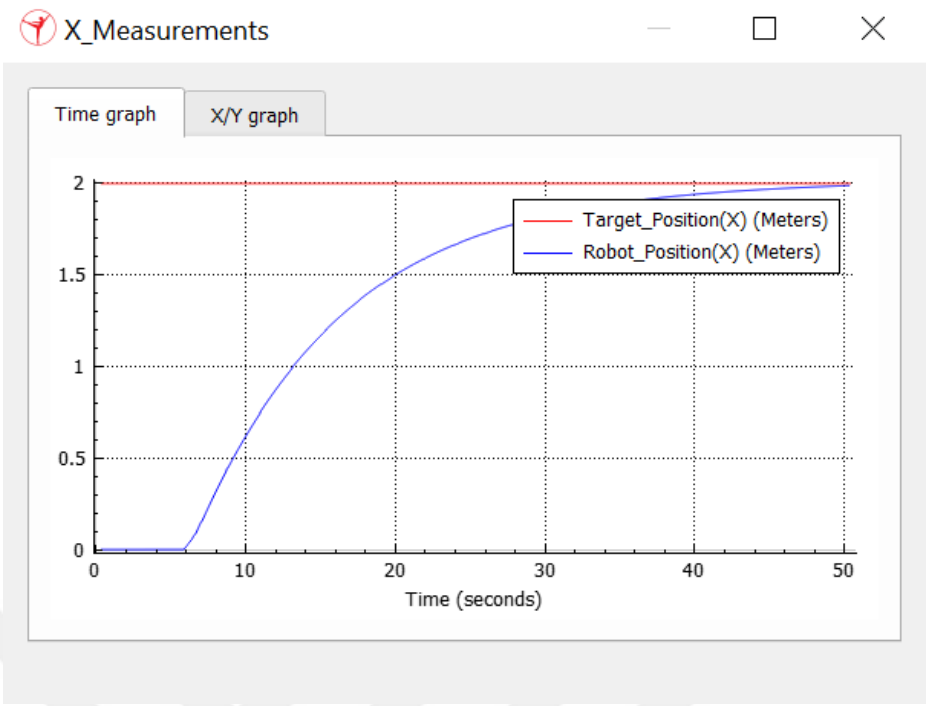


Figure 5.7. The Omni-wheels robot X-coordinates of scenario 2.

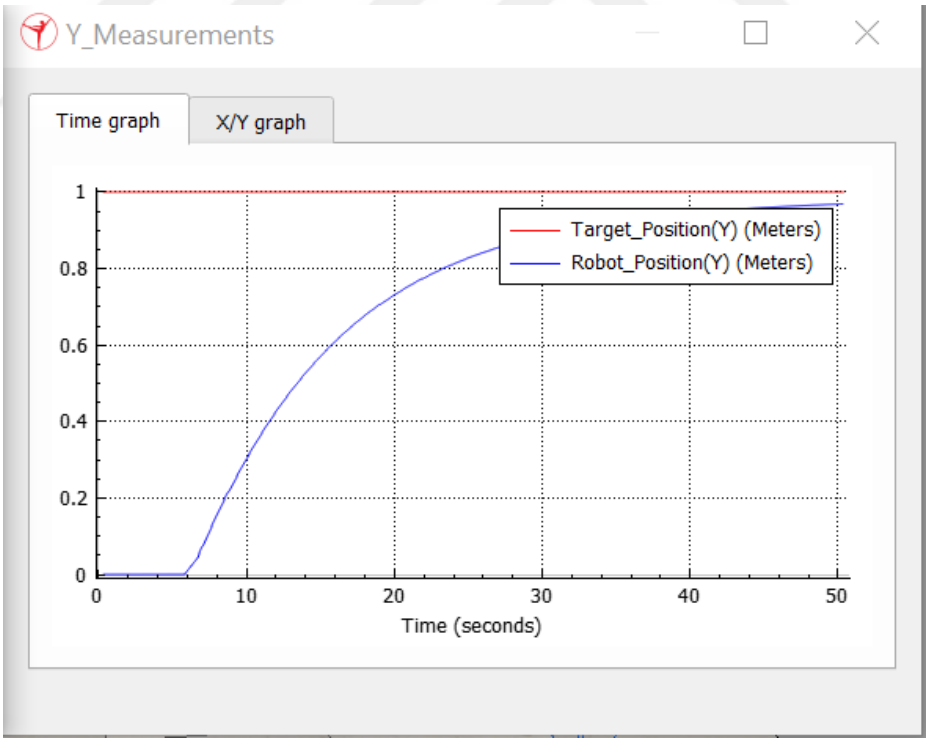


Figure 5.8. The Omni-wheels robot Y-coordinates of scenario 2.

6. CONCLUSIONS

In this work, the indoor localization of the Omni-wheels robot was simulated using the CoppeliaSim robot simulator. The indoor localization of the robot was achieved by applying the trilateration method. This method depends on identifying color-coded beacons to obtain their position with respect to the global frame and calculating the distances between each beacon and the robot. These beacons have been designed in circular shapes so that the detection of them can be done easily from any robot orientation. Two scenarios were examined to prove the capability of the proposed method for robot indoor localization. In both scenarios, the simulation results show that the robot rotates around itself to achieve the desired angle, then moves towards the target pose. The designed CCs beacons have enabled the Omni-wheels robot to estimate its position and reach the desired target.

The contributions of this work include: i) unlike other methods, the approach in this paper uses passive beacons of circular shapes, which do not require a power source and are easy to build and scale (more beacons can be generated by increasing the number of rings forming them); ii) one sensor (depth camera) is all that is required to accurately localize the robot in an indoor environment; and iii) the effectiveness of the proposed localization method is verified using the realistic CoppeliaSim robot simulator.

REFERENCES

[1] Fabregas, E., Farias, G., Peralta, E., Vargas, H. & Dormido, S., (2016). Teaching control in mobile robotics with V-REP and a Khepera IV library.Conference on control applications (CCA).IEEE.pp.821-826.

[2] Ilon, B.E., (1975). Wheels for a course stable self-propelling vehicle movable in any desired direction on the ground or some other base.

[3] Doroftei, I., Grosu, V. & Spinu, V., (2007). Omnidirectional mobile robot–design and implementation. INTECH Open Access Publisher. I-Tech, Vienna, Austria. pp.544.

[4] Siegwart, R., Nourbakhsh, I.R., and Scaramuzza, D., (2011). Introduction to autonomous mobile robots. Intelligent Robotics and Autonomous Agents series,.pp. 472

[5] Campion, G., Bastin, G., and D'andrea, N. B., (2011). Structural properties and classification on kinematic and dynamic models of wheeled mobile robots. Nelineinaya Dinamika [Russian Journal of Nonlinear Dynamics], 7(4), (pp. 733-769).

[6] Pace, S., Frost, G., Lachow, I., Frelinger, D., and Fossum, D., (1995). The global positioning system: assessing national policies.pp. 12-245.

[7] Ladd, A.M., Bekris, K.E., Rudys, A.P., Wallach, D.S., and Kavraki, L.E., (2004). On the feasibility of using wireless ethernet for indoor localisation. IEEE Transactions on Robotics and Automation, 20(3), pp. 555-559.

[8] Jensfelt, P., (2001). Approaches to mobile robot localisation in indoor environments.Citeseer.pp. 19-222

[9] Barshan, B., and Durrant, W.H.F., (1995). Inertial navigation systems for mobile robots. IEEE Transactions on Robotics and Automation, 11(3), pp. 328-342.

[10] Bittle, O., and Blaich, M., (2011). Mobile robot localisation using beacons and the Kalman filter technique for the Eurobot competition, International Conference on Research and Education in Robotics, Springer, pp. 55-67.

[11] Bekris, K.E., Argyros, A.A. and Kavraki, L.E., (2004). Angle-based methods for mobile robot navigation: Reaching the entire plane, Robotics and Automation. IEEE International Conference on 2004, pp. 2373-2378.

[12] Huosheng, H., and Dongbing, G., (2000). Landmark-based navigation of industrial mobile robots. Industrial Robot: An International Journal, 27(6), pp. 458-467.

[13] Chuang, L., (2012). Object localisation strategy for a mobile robot using RFID.1(1). 5.

[14] Gueaieb, W., and Miah, M.S., (2008). An intelligent mobile robot navigation technique using RFID technology. IEEE Transactions on Instrumentation and Measurement, 57(9), (pp. 1908-1917).

[15] Kostomins, A., and Osadcuks, V., (2014). Infrared active beacon application evaluation for mobile robot localisation in agriculture. Engineering for Rural Development (Latvia).pp. 373-378

[16] Beom, H.R. and Cho, H.S., (1995). Mobile robot localisation using a single rotating sonar and two passive cylindrical beacons. Robotica, 13(03), pp. 243-252.

[17] Kleeman, L., (1992). Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning, Robotics and Automation,IEEE International Conference on 1992, pp. 2582-2587.

[18] Kresja, J., and Vechet, S., (2012). Infrared beacons based localisation of mobile robot. Elektronika ir Elektrotechnika, 117(1), pp. 17-22.

[19] Yamamoto, Y., Pirjanian, P., Munich, M., Dibernardo, E., Goncalves, L., Ostrowski, J. and Karlsson, N., (2005). Optical sensing for robot perception and localization, IEEE Workshop on Advanced Robotics and its Social Impacts, 2005. pp. 14-17.IEEE.

[20] Carrick, D., John, L., Daniela, R., and Seth, T., (2006). Passive Mobile Robot Localization within a Fixed Beacon Field. WAFR (Seventh International Workshop on the Algorithmic Foundations of Robotics, pp.1-16

[21] Eric, N., Jacques, G., Mohammed, T., Umar, I., and Aboelmagd, N., (2010). Enhanced Mobile Robot Outdoor Localization Using INS/GPS Integration. IEEE.pp. 127-132

[22] Peralta, E., Fabregas, E., Farias, G., Vargas, H., & Dormido, S. (2016). Development of a Khepera IV Library for the V-REP Simulator. IFAC-PapersOnLine, 49(6), pp. 81-86.

[23] URL1 V-REP simülatör: http://www.coppeliarobotics.com/helpfiles/en/visionsensor.htm, 7 January 2021.

[24] Muir, P.F., and Neuman, C.P., (1990). Kinematic modeling for feedback control of an omnidirectional wheeled mobile robot. Autonomous robot vehicles. Springer, pp. 25-31.

[25] AL-Ammari, A.S. and Ahmed, I., (2010). Control of Omni-Directional Mobile Robot Motion. Baghdad: Al-Khwarizmi Engineering Journal.6(4). pp. 1-9.

[26] Francisco R., Francisco V. and Carlos L.A.,(2019). A review of mobile robots: Concepts, methods, theoretical framework, and applications. International Journal of Advanced Robotic Systems.16(2).pp. 1-22.

[27] Roberto, I., Luiz, H. F., and Waldemar, C., (2003). Lua 5.0 Reference Manual.pp. 11-38

[28] Dorf, R.C. and Bishop, R.H., (2011). Modern control systems. Prentice-Hall.pp.8-723

[29] Kenny A. Q. C., Junior A. R. S., Felix M. E., Valdir G. Jr., Marco H.T. and Adriano A.G.S.(2017). A comparison between a simulated and a real mobile robot path tracking application using V-REP. Simposio Brasileiro de Automacao Inteligente. pp. 2026-2031

[30] Felipe, P. C., Joao, A. F., and Andre S. O., (2020). Integrating mobile robot navigation control, visualobject detection and manipulation using ROS and V-REP. Conference: Congresso Brasileiro de Inteligência Computacional. pp.1-5

[31] Mohanad, N. N., Mohammed, Q., and Omar, Y. I., (2021). Landmarks exploration algorithm for mobile robot indoor localization using vision sensor. Journal of Engineering Science and Technology.16 (4).pp. 3165 – 3184.

[32] Vladimir, P., and Jelena P.B., (2016). Towards Real-Time Blob Detection in Large Images with Reduced Memory Cost. Proceedings of 3rd International Conference on Electrical, Electronic and Computing Engineering.pp. 1-6

[33] Alatise, M.B.; and Hancke, G.P. (2017). Pose estimation of a mobile robot based on fusion of IMU data and vision data using an extended Kalman filter. *Sensors*, 17(10), pp. 1-22.

[34] Pablo, C., Miguel, V., David, C., Luis, M. and Manuel, B., (2016). Indoor Robot Positioning using an Enhanced Trilateration Algorithm. Engineering and Industrial Development Center.13(3).1-8

[35] Ranjit, B., Sajal, M., and Dr. Samiran, M., (2015). Motion Analysis of A Mobile Robot With Three Omni-Directional Wheels. IJISET - International Journal of Innovative Science, Engineering & Technology, 2(11).pp.644-648

APPENDICES

APPENDIX- 1: DERIVATIVE OF TRILATERATION METHOD

$$d_1^2 = (X - X_1)^2 + (Y - Y_1)^2 \quad \text{(A1.1)}$$

$$d_2^2 = (X - X_2)^2 + (Y - Y_2)^2 \quad \text{(A1.2)}$$

$$d_3^2 = (X - X_3)^2 + (Y - Y_3)^2 \quad \text{(A1.3)}$$

After subtracting equation (A1.2) from (A1.1) we obtain:

$$-2XX_1 + 2XX_2 + {X_1}^2 - {X_2}^2 - 2YY_1 + 2YY_2 + {Y_1}^2 - {Y_2}^2 = d_1^2 - d_2^2$$

$$X(-2X_1 + 2X_2) + Y(-2Y_1 + 2Y_2) = d_1^2 - d_2^2 - {X_1}^2 + {X_2}^2 - {Y_1}^2 + {Y_2}^2$$

Where:

$$A = -2X_1 + 2X_2$$

$$B = -2Y_1 + 2Y_2$$

$$C = d_1^2 - d_2^2 - {X_1}^2 + {X_2}^2 - {Y_1}^2 + {Y_2}^2$$

This yields to a new equation:

$$AX + BY = C \quad \text{(A1.4)}$$

And by subtracting equation (A1.3) from (A1.2) we get:

$$X(-2X_2 + 2X_3) + Y(-2Y_2 + 2Y_3) = d_2^2 - d_3^2 - {X_2}^2 + {X_3}^2 - {Y_2}^2 + {Y_3}^2$$

Where :

$$D = -2X_2 + 2X_3$$

$$E = -2Y_2 + 2Y_3$$

$$F = d_2^2 - d_3^2 - {X_2}^2 + {X_3}^2 - {Y_2}^2 + {Y_3}^2$$

A new equation has been formed:

$$DX + EY = F \quad \text{(A1.5)}$$

Then by multiplying (A1.4) with (E) and (A1.5) with (B) we obtain:

$$AEX + BEY = CE \quad \text{(A1.6)}$$

$$DBX + EBY = FB \tag{A1.7}$$

After subtracting (A1.7) from (A1.6) we obtain:

$$X = \frac{CE - FB}{AE - DB} \tag{A1.8}$$

by multiplying (A1.4) with (D) and (A1.5) with (A) we obtain:

$$ADX + BDY = CD \tag{A1.9}$$

$$DAX + EAY = FA \tag{A1.10}$$

After subtracting (A1.10) from (A1.9) we obtain:

$$Y = \frac{CD - FA}{DB - AE} \tag{A1.11}$$

APPENDIX- 2: COPPELIASIM PROGRAM AND FUNCTIONS

```
function sysCall_init()
    omnipads={-1,-1,-1}
    omnipads[1]=sim.getObjectHandle('OmniWheel1')
    omnipads[2]=sim.getObjectHandle('OmniWheel2')
    omnipads[3]=sim.getObjectHandle('OmniWheel3')
    omniRob=sim.getObjectHandle('omniRob')
    vs=sim.getObjectHandle('VS')
    L=0.093
    r=0.017635
    d=0
    b121=sim.getObjectHandle('121')
    b123=sim.getObjectHandle('123')
    b131=sim.getObjectHandle('131')
    b132=sim.getObjectHandle('132')
    b212=sim.getObjectHandle('212')
    b213=sim.getObjectHandle('213')
    b231=sim.getObjectHandle('231')
    b232=sim.getObjectHandle('232')
    b312=sim.getObjectHandle('312')
    b313=sim.getObjectHandle('313')
    b321=sim.getObjectHandle('321')
    b323=sim.getObjectHandle('323')
   sizeOf_RGB={} BeaconNo_RGB=0 RGB_ArrayCount=0
sizeOf_RRG={} BeaconNo_RRG=0 RRG_ArrayCount=0
sizeOf_RRB={} BeaconNo_RRB=0 RRB_ArrayCount=0
sizeOf_GGR={} BeaconNo_GGR=0 GGR_ArrayCount=0
sizeOf_BBR={} BeaconNo_BBR=0 BBR_ArrayCount=0
sizeOf_GGB={} BeaconNo_GGB=0 GGB_ArrayCount=0
sizeOf_BBG={} BeaconNo_BBG=0 BBG_ArrayCount=0
BeaconTotalR=0
arrayCount=0
BeaconNo=0
BeaconPosition={}
  ---Target Position and Orientation----------
xt=-3
yt=-1
thetatarget=0.321-0.52
end
```

Figure A2.1. Screenshot for CoppeliaSim Program.

```
function sysCall_actuation()
SensorColorCodes={}
   r2,p1,red,green,blue = sim.readVisionSensor(vs)
   SensorColorCodes= codeDiscovery(r2,p1,red,green,blue)
thetaRobot=sim.getObjectOrientation(omniRob,-1)[3]
err=thetatarget-thetaRobot
        if (err<=0.01) and (D_err > 0.023) then
           D_err=math.sqrt(math.pow((xt-Xr),2)+ math.pow((yt-Yr),2))
           Xr_dot=-0.1*D_err
           Yr_dot=0
          Theta_dot=0
       else
           Xr_dot=0
           Yr_dot=0
           Theta_dot=err*1
       end
sim.setJointTargetVelocity(omnipads[1], (1/r)*( Xr_dot*math.sqrt(3)/2 -0.5*Yr_dot - L*Theta_dot))
sim.setJointTargetVelocity(omnipads[2], (1/r)*(                          Yr_dot - L*Theta_dot))
sim.setJointTargetVelocity(omnipads[3], (1/r)*(-Xr_dot*math.sqrt(3)/2 -0.5*Yr_dot - L*Theta_dot))
   taken={ {},{},{}}

takcount=1
for u=1 ,3 do
    for b=1 ,#BeaconPosition do
       if(SensorColorCodes[u]==BeaconPosition[b][1]) then
            taken[takcount][1]=BeaconPosition[b][2][1]
           taken[takcount][2]=BeaconPosition[b][2][2]
        takcount=takcount+1
       end
    end
end
takcount=1
xb1=taken[1][1]
xb2=taken[2][1]
xb3=taken[3][1]
yb1=taken[1][2]
yb2=taken[2][2]
yb3=taken[3][2]
XO=sim.getObjectPosition(omniRob,-1)[1]
YO=sim.getObjectPosition(omniRob,-1)[2]
d1=math.sqrt(math.pow((XO-xb1),2)+ math.pow((YO-yb1),2))
d2=math.sqrt(math.pow((XO-xb2),2)+ math.pow((YO-yb2),2))
d3=math.sqrt(math.pow((XO-xb3),2)+ math.pow((YO-yb3),2))
A=(-2*xb1)+(2*xb2)
B=(-2*yb1)+(2*yb2)
C=math.pow(d1,2)-math.pow(d2,2)-math.pow(xb1,2)+math.pow(xb2,2)-math.pow(yb1,2)+math.pow(yb2,2)
D=(-2*xb2)+(2*xb3)
E=(-2*yb2)+(2*yb3)
F=math.pow(d2,2)-math.pow(d3,2)-math.pow(xb2,2)+math.pow(xb3,2)-math.pow(yb2,2)+math.pow(yb3,2)
Xr=((C*E)-(F*B))/((E*A)-(B*D))
Yr=((C*D)-(A*F))/((B*D)-(A*E))
end
```

Figure A2.2. Screenshot for CoppeliaSim Program.

```
----------FUNCTIONS---
function codeDetection_RGB(sizeOfBlobs,BeaconTotal)
colorCode={}
count=0
    for i=1,BeaconTotal do
        SR=sizeOfBlobs[i+count]
        SG=sizeOfBlobs[i+1+count]
        SB=sizeOfBlobs[i+2+count]

        if(SR>SG)and(SR>SB)and(SG>SB) then colorCode[i]=123
        elseif(SR>SB)and(SR>SG)and(SB>SG) then colorCode[i]=132
        elseif(SG>SR)and(SG>SB)and(SR>SB) then colorCode[i]=213
        elseif(SG>SB)and(SG>SR)and(SB>SR) then colorCode[i]=231
        elseif(SB>SR)and(SB>SG)and(SR>SG) then colorCode[i]=312
        elseif (SB>SG)and(SB>SR)and(SG>SR) then colorCode[i]=321
        else return 'no mact'
        end
        count=count+2
    end
    return colorCode
end
-------RRG-Beacon---------
function codeDetection_RRG(sizeOfBlobs,BeaconTotal)
        colorCode={}
        count=0
        for i=1,BeaconTotal do
        SR1=sizeOfBlobs[i+count]
        SR2=sizeOfBlobs[i+1+count]
        SG=sizeOfBlobs[i+2+count]
        if(SR1>SG)and(SR1>SR2)and(SG>SR2) then colorCode[i]=121
        else return 'no mach'
        end
        count=count+2
        end
    return colorCode
end
----------RRB-Beacon---------
function codeDetection_RRB(sizeOfBlobs,BeaconTotal)
        colorCode={}
        count=0
        for i=1,BeaconTotal do
        SR1=sizeOfBlobs[i+count]
        SR2=sizeOfBlobs[i+1+count]
        SB=sizeOfBlobs[i+2+count]
         if(SR1>SB)and(SR1>SR2)and(SB>SR2) then colorCode[i]=131
        else return 'no mach'
        end
        count=count+2
        end
    return colorCode
```

Figure A2.3. Screenshot for CoppeliaSim Program.

```
function codeDetection_GGR(sizeOfBlobs,BeaconTotal)
        colorCode={}
        count=0
        for i=1,BeaconTotal do
        SG1=sizeOfBlobs[i+count]
        SG2=sizeOfBlobs[i+1+count]
        SR=sizeOfBlobs[i+2+count]
              if(SG1>SR)and(SG1>SG2)and(SR>SG2) then colorCode[i]=212
        else return 'no mach'
        end
        count=count+2
       end
    return colorCode
end
 ------BBR-Beacon---------
function codeDetection_BBR(sizeOfBlobs,BeaconTotal)
        colorCode={}
        count=0
        for i=1,BeaconTotal do
        SB1=sizeOfBlobs[i+count]
        SB2=sizeOfBlobs[i+1+count]
        SR=sizeOfBlobs[i+2+count]

        if(SB1>SR)and(SB1>SB2)and(SR>SB2) then colorCode[i]=313
        else return 'no mach'
        end
        count=count+2
        end
    return colorCode
end
 ------GGB-Beacon---------
function codeDetection_GGB(sizeOfBlobs,BeaconTotal)
        colorCode={}
        count=0
        for i=1,BeaconTotal do
        SG1=sizeOfBlobs[i+count]
        SG2=sizeOfBlobs[i+1+count]
        SB=sizeOfBlobs[i+2+count]
              if(SG1>SB)and(SG1>SG2)and(SB>SG2) then colorCode[i]=232
        else return 'no mach'
        end
        count=count+2
        end
    return colorCode
end
```

Figure A2.4. Screenshot for CoppeliaSim Program.

```
        end
    return colorCode
end
-------BBG-Beacon---------
function codeDetection_BBG(sizeOfBlobs,BeaconTotal)
        colorCode={}
        count=0
        for i=1,BeaconTotal do
        SB1=sizeOfBlobs[i+count]
        SB2=sizeOfBlobs[i+1+count]
        SG=sizeOfBlobs[i+2+count]
         if(SB1>SG)and(SB1>SB2)and(SG>SB2) then colorCode[i]=323
        else return 'no mach'
        end
        count=count+2
       end
    return colorCode
end

function doubleBlobDetector(Flag1,Flag2,BeaconNo,sizeOf2DoubleColorArray,
arrayCount,sizeOfBlob1,sizeOfBlob2,sizeOfBlob3)
                if (Flag1==true) and (Flag2==true) then

                   sizeOf2DoubleColorArray[1+arrayCount]=sizeOfBlob1
                    sizeOf2DoubleColorArray[2+arrayCount]=sizeOfBlob2
                    sizeOf2DoubleColorArray[3+arrayCount]=sizeOfBlob3
                    arrayCount=arrayCount+3
                    BeaconNo=BeaconNo+1
                        Flag1=false
                        Flag2=false
                end
return sizeOf2DoubleColorArray,BeaconNo,arrayCount
end
    function codeDiscovery (r,pl,red,green,blue)
    redFlag=false
    greenFlag=false
    blueFlag=false
    Red_RedFlag=false
    green_greenFlag=false
     Green_double=false
     Blue_double=false
     Green_blue_double=false
      Blue_green_double=false
  flag_Check_GG=1 GG_Size={}
flag_Check_BB=1 BB_Size={}
flag_Check_GGB=1 GGB_Size={}
flag_Check_BBG=1 BBG_Size={}
F for i=1,red[1] do
flag_Check_GG=1
flag_Check_BB=1
        blobSizeRed=red[3+red[2]*(i-1)+0]*1000
------RED-GREEN-Beacon---------
                    for j=1,green[1] do
dRed_Green=math.sqrt(math.pow(red[3+red[2]*(i-1)+2]-green[3+green[2]*(j-1)+2],2)+
```

Figure A2.5. Screenshot for CoppeliaSim Program.

```
dRed_Green=math.sqrt(math.pow(red[3+red[2]*(i-1)+2]-green[3+green[2]*(j-1)+2],2)+
math.pow(red[3+red[2]*(i-1)+3]-green[3+green[2]*(j-1)+3],2))
                        if ((dRed_Green)<0.004) then
                           blobSizeGreen=green[3+green[2]*(j-1)+0]*1000
                           greenFlag=true
                           GG_Size[flag_Check_GG]=blobSizeGreen
                           flag_Check_GG=flag_Check_GG+1
                                   if  (flag_Check_GG==3) then
                                         Green_double=true
                                           break
                        end
                    end
----RED-BLUE-Beacon---------

                    for k=1,blue[1] do
                         dRed_blue= math.sqrt(math.pow(red[3+red[2]*(i-1)+2]-blue[3+blue[2]*(k-1)+2],2)+math.pow(red[3+red[2]*(i-1)+3]-blue[3+blue[2]*(k-1)+3],2))
                        if ((dRed_blue)<0.004) then
                            blobSizeBlue=blue[3+blue[2]*(k-1)+0]*1000
                            blueFlag=true
                         BB_Size[flag_Check_BB]=blobSizeBlue
                           flag_Check_BB=flag_Check_BB+1
                                   if  (flag_Check_BB==3) then
                                         Blue_double=true
                                        break
                                   end
                          end
                              end
---RED-RED-Beacon---------
                    for l=1+i,red[1] do
                         dRed_Red= math.sqrt(math.pow(red[3+red[2]*(i-1)+2]-red[3+red[2]*(l-1)+2],2)+math.pow(red[3+red[2]*(i-1)+3]-red[3+red[2]*(l-1)+3],2))
                        if ((dRed_Red)<0.004) then
                        blobSizeRed_Red=red[3+red[2]*(l-1)+0]*1000
                            Red_RedFlag=true
                                                end
                                 end
-------------------------------------
sizeOf_RGB,BeaconNo_RGB,RGB_ArrayCount=doubleBlobDetector(greenFlag,blueFlag,BeaconNo_RGB,sizeOf_RGB,RGB_ArrayCount,blobSizeRed,blobSizeGreen,blobSizeBlue)
sizeOf_RRG,BeaconNo_RRG,RRG_ArrayCount=doubleBlobDetector(Red_RedFlag,greenFlag,BeaconNo_RRG,sizeOf_RRG,RRG_ArrayCount,blobSizeRed,blobSizeRed_Red,blobSizeGreen)
sizeOf_RRB,BeaconNo_RRB,RRB_ArrayCount=doubleBlobDetector(Red_RedFlag,blueFlag,BeaconNo_RRB,sizeOf_RRB,RRB_ArrayCount,blobSizeRed,blobSizeRed_Red,blobSizeBlue)
sizeOf_GGR,BeaconNo_GGR,GGR_ArrayCount=doubleBlobDetector(Green_double,greenFlag,BeaconNo_GGR,sizeOf_GGR,GGR_ArrayCount,GG_Size[1],GG_Size[2],blobSizeRed)
sizeOf_BBR,BeaconNo_BBR,BBR_ArrayCount=doubleBlobDetector(Blue_double,blueFlag,BeaconNo_BBR,sizeOf_BBR,BBR_ArrayCount,BB_Size[1],BB_Size[2],blobSizeRed)
   greenFlag=false
   blueFlag=false
   Red_RedFlag=false
   green_greenFlag=false
   Green_double=false
   Blue_double=false

    end
------   END OF MAIN FOR    --------
-------GREEN-GREEN-BLUE-Beacon---------
   for m=1,blue[1] do
flag_Check_GGB=1
   blueFlag=false
   Red_RedFlag=false
   green_greenFlag=false
   Green_double=false
   Blue_double=false

    end
------   END OF MAIN FOR    --------
-------GREEN-GREEN-BLUE-Beacon---------
   for m=1,blue[1] do
flag_Check_GGB=1
       blobSizeBlue=blue[3+blue[2]*(m-1)+0]*1000
       for n=1,green[1] do
           dBlue_Green=math.sqrt(math.pow(blue[3+blue[2]*(m-1)+2]-green[3+green[2]*(n-1)+2],2)+math.pow(blue[3+blue[2]*(m-1)+3]-green[3+green[2]*(n-1)+3],2))

                       if ((dBlue_Green)<0.004) then
                          blobSizeGreen=green[3+green[2]*(n-1)+0]*1000
                          blue_Green_singleFlag=true

                          GGB_Size[flag_Check_GGB]=blobSizeGreen
                          flag_Check_GGB=flag_Check_GGB+1
                                  if  (flag_Check_GGB==3) then

                                          Green_blue_double=true
                                             break
                                  end
                   end
         end
sizeOf_GGB,BeaconNo_GGB,GGB_ArrayCount=doubleBlobDetector(Green_blue_double,Green_blue_double,BeaconNo_GGB,sizeOf_GGB,GGB_ArrayCount,GGB_Size[1],GGB_Size[2],blobSizeBlue)
Green_blue_double=false
    end
--------BLUE-BLUE-GREEN-Beacon---------
for s=1,green[1] do
flag_Check_BBG=1
       blobSizeGreen=green[3+green[2]*(s-1)+0]*1000
       for z=1,blue[1] do
           dBlue_Green=math.sqrt(math.pow(green[3+green[2]*(s-1)+2]-blue[3+blue[2]*(z-1)+2],2)+math.pow(green[3+green[2]*(s-1)+3]-blue[3+blue[2]*(z-1)+3],2))
                       if ((dBlue_Green)<0.004) then
                           blobSizeBlue=blue[3+blue[2]*(z-1)+0]*1000
                           blue_Green_singleFlag=true
                           BBG_Size[flag_Check_BBG]=blobSizeBlue
                          flag_Check_BBG=flag_Check_BBG+1
                                  if  (flag_Check_BBG==3) then
                                          Blue_green_double=true
                                             break
                                  end
                    end
         end
sizeOf_BBG,BeaconNo_BBG,BBG_ArrayCount=doubleBlobDetector(Blue_green_double,Blue_green_double,BeaconNo_BBG,sizeOf_BBG,BBG_ArrayCount,BBG_Size[1],BBG_Size[2],blobSizeGreen)
Blue_green_double=false
    end
---------------------------------
codes_RGB=codeDetection_RGB(sizeOf_RGB,BeaconNo_RGB)
codes_RRG=codeDetection_RRG(sizeOf_RRG,BeaconNo_RRG)
```

Figure A2.6. Screenshot for CoppeliaSim Program.

```
codes_RRB=codeDetection_RRB(sizeOf_RRB,BeaconNo_RRB)
codes_GGR=codeDetection_GGR(sizeOf_GGR,BeaconNo_GGR)
codes_BBR=codeDetection_BBR(sizeOf_BBR,BeaconNo_BBR)
codes_GGB=codeDetection_GGB(sizeOf_GGB,BeaconNo_GGB)
codes_BBG=codeDetection_BBG(sizeOf_BBG,BeaconNo_BBG)

--------------Set To ZEROs-----------
BeaconTotalR=0 BeaconNo_RGB=0 RGB_ArrayCount=0
sizeOf_RRG={} BeaconNo_RRG=0 RRG_ArrayCount=0
sizeOf_RRB={} BeaconNo_RRB=0 RRB_ArrayCount=0
sizeOf_GGR={} BeaconNo_GGR=0 GGR_ArrayCount=0
sizeOf_BBR={} BeaconNo_BBR=0 BBR_ArrayCount=0
sizeOf_GGB={} BeaconNo_GGB=0 GGB_ArrayCount=0
sizeOf_BBG={} BeaconNo_BBG=0 BBG_ArrayCount=0

flag_Check_GG=1 GG_Size={}
flag_Check_BB=1 BB_Size={}
flag_Check_GGB=1 GGB_Size={}
flag_Check_BBG=1 BBG_Size={}
-------------------------
fusedArray = {}
n=0
for k,v in ipairs(codes_RGB) do n=n+1 ; fusedArray[n] = v end
for k,v in ipairs(codes_RRG) do n=n+1 ; fusedArray[n] = v end
for k,v in ipairs(codes_RRB) do n=n+1 ; fusedArray[n] = v end
for k,v in ipairs(codes_GGR) do n=n+1 ; fusedArray[n] = v end
for k,v in ipairs(codes_BBR) do n=n+1 ; fusedArray[n] = v end
for k,v in ipairs(codes_GGB) do n=n+1 ; fusedArray[n] = v end
for k,v in ipairs(codes_BBG) do n=n+1 ; fusedArray[n] = v end
 return fusedArray
    end
```

Figure A2.7. Screenshot for CoppeliaSim Program.

CURRICULUM VITAE

Mawj MOHAMMED BASHEER

PERSONAL INFORMATION

[redacted]