Photocopy and Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the UniversityLibrarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

Object Detection, Localization and Navigation Strategy for Obstacle Avoidance Applied to Autonomous Wheelchair Driving

by

Nusrat Farheen

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in the Department of Mechanical Engineering

Idaho State University

Spring 2022

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Nusrat Farheen find it satisfactory and recommend that it be accepted.

_____

Dr. Marco P. Schoen, Major Advisor

_____

Dr. Kenneth Bosworth, Committee Member

_____

Dr. Nancy Devine, Graduate Faculty Representative

Acknowledgements

First and foremost, I want to express my gratitude to Dr. Marco Schoen, my primary advisor, for his aid and direction in finishing my thesis. I would like to express my gratitude for his patience and kindness. This research would not have been attainable without his expertise, enthusiasm, and assistance. I appreciate the Career Path Internships and Graduate Teaching Assistantships provided by Dr. Bosworth and Dr. Alba Perez. Thank you for being an amazing mentor and believing in me. I want to thank Dr. Bosworth for taking the time to teach me math and for his willingness to give me time for any sort of discussions, also for supporting me every time with his recommendations associated with scholarships applications. Thank you, Dr. Nancy Devine, for agreeing to be in my committee, and giving your valuable time. I want to express my appreciation to Ellen and Laura for their help along the way. Thank you to everyone at MCERC for all of your support; it has been a pleasure sharing the lab with you. I appreciate my friends and family's patience and support during my studies. My spouse Golam deserves special thanks for assisting me, for his continuous support, and for his presence in the ERC lab. Thank you very much!

Object Detection, Localization and Navigation Strategy for Obstacle Avoidance Applied to Autonomous Wheelchair Driving

Thesis Abstract – Idaho State University (2022)

The primary aim of this study was to develop machine learning or deep-learning aided procedures along with scientific investigations that enhances the capability of a commercial non-autonomous wheelchair towards autonomy. The thesis addresses the computer vision work for obstacle detection and localization applied to an autonomous wheelchair operation. The computer vision tasks including the depth image classification are accommodated in a small form factored and resource constraint computers such as Raspberry Pie and Google Coral. The tasks and strategies also include classifying the images using a pretrained model (TensorFlow lite), detecting and measure the degree of obstacle avoidance by pairing color (RGB) image classification with depth images. The thesis also offers approaches for indoor localization applicable for the autonomous wheelchairdevelopment. The objective has been further extended to develop a simulation platform for autonomous wheelchair driving where navigation and path mapping construction algorithm evaluations are visually offered using MATLAB®. In addition, the thesis includes research and project contributions prior to the change of thesis subject to autonomous wheelchair development. These contributions are addressed as the additional works which includes (1) the initial work on error determination in motion capture process using VICON and (2) a wafer alignment fault detection process using image processing.

Keywords: machine learning, computer vision, autonomous wheel-chair, navigation, mobile robot, depth image, obstacle avoidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Literature Review

A Powered wheelchair has been a crucial tool to help regain lost mobility applied. The sole purpose of a powered wheelchair is to provide mobility assistance without needing an additional human interface [1]. Removal of dependency on others is tied to mental health and speedy recovery [2]. Hence, for more than a hundred years, powered wheelchairs received innovative attention from scientific and engineering communities. The evolution of the powered wheelchair aligned to the objective of making the power wheelchairs as intuitive to operate as possible while keeping the cost minimum. The cost includes energy consumption, satisfactory performance metrics, and of course financial components. During the evolution of wheelchairs, there were several phases of development. Most of those developments involve minimizing power consumption, generating comfort, simpler operation procedures and making the wheelchairs affordable through new technologies [3]. In recent decades a paradigm shift is observed by the tremendous inclusion of data and machine learning techniques. As the semiconductor industries push low cost, low power and smaller but powerful computers, more and more data driven portable system are gaining attention [4]. One of the key research questions included in the data driven solution to powered wheelchairs is how the safety features can be enhanced. There is a need to discover how far an existing powered wheelchair can be stretched from manual operation to assistive technologies to full autonomy.

The benefits of full autonomy in powered wheelchair starts with including wider range of users who seek independence from external physical assistance [5]. Moreover, autonomous wheelchairs

add layers of security and safety in operation by managing the data it collects from the runtime. In search of full autonomy, research effort has been put to strengthen the feedback process for accurate assessment of obstacle detection, identification of hazardous route along with indoor localization and path planning. In Gao, Sands & Speltzer (2010) [6], discussed use of LiDAR to generate a map of the autonomous wheelchair operation environment. LiDAR is proven to be an effective tool in autonomous vehicle due to features providing multiple obstacle information in real-time. Also, LiDAR can operate in the dark. However, it lacks understanding of the objects around it due to incapability of differentiating colors and low resolution. Concerns indicated with the LiDAR technology in autonomous wheelchairs can be overcome by fusing with the other sensors such as RGB cameras. However, that does not remove the fact that LiDAR is relatively a large and heavy sensor that consumes relatively more power and is expensive [7]. RGB cameras on the other hand, provide information for object identification and can be implemented by fusing other proximity sensors such as ultrasonic sensors, IR, etc. As the number of sensors grows, latency adds up to overall in situ monitoring and management of cross data.

Another critical aspect of research in autonomous wheelchairs is accurate assessment of location of the wheelchair. Indoor localization is particularly challenging due to elevated obstacle counts and dedicated system requirement to obtain accurate coordinates [8]. There are solutions proposed in Lankenau, Röfer, Krieg-Brückner, (2003) [9], such as a network of IR beams to obtain the positions of autonomous mobile robots. However, the system with an IR network often creates large error which is infeasible in sensitive cases such as the autonomy of wheelchairs. However, all of these aforementioned research efforts indicate the current research state for autonomous wheelchairs is analogous to autonomy of mobile robots. Therefore, autonomous powered wheelchairs can be treated as a branch of autonomous mobile robots [10].

1.2 Autonomous wheelchair (AWC) as a special purpose mobile robot

Robotics have greatly shaped many tasks around us that once needed human interference explicitly [11]. Autonomous robots bring significant improvement in daily activities, production capabilities and services by having an increased accuracy and pace over nonautomated methods [12]. Most of these autonomous actions are witnessed in chain productions, warehouse management, and packaging. However, the class of robots found in research is spread across a wide range of applications from humanoid to surgical robots. Mobile robots have been a popular study for a long time. Research in mobile robots has grown to an extent that this field offers an umbrella of subfields. Mobile robots are found in applications requiring some amount of artificial intelligence such as collaborative robots, swarm intelligence-based robots, autonomous aerial vehicles, autonomous ground vehicles, autonomous cleaning products, autonomous wheelchairs, etc., [13]. The thesis addresses a specific task associated with an autonomous wheelchair (AWC). Just as any other autonomous mobile robot, an AWC is composed of actuation, feedback from the environment, and a processing facility. The actuation in the thesis points to motor action corresponding to locomotion. The feedback includes all forms of sensors and communication either direct or derived from the environment and finally, the processing indicates computations involved in decision making and control instructions.

1.3 Key areas of AWC addressed by researchers

The application of an AWC is a sensitive implementation whether in the medical care facility or in a home. The use of AWC equipment often involves people with physical disabilities. Hence, safety and comfort carry higher priority over speed and accuracy. Therefore, the development of

an autonomous wheelchair involves four streams of research which are as follows: (1) Development of appropriate localization system depending on the site where the system is going to be running, [14]; (2) A feedback system that gathers data from the environment to enhance safe and secure operation, [15]; (3) Navigation strategies to perform locomotion without compromising safety and comfort including mapping, monitoring, and simulation for the performance analysis and future improvements, [16]; and (4) a control system and an associated mechanism involving manipulation of the velocity and the direction of the AWC, [17].

2. Critical Tasks and Strategies

2.1 Localization and Orientation

Localization and orientation tasks of an AWC compiles indoor/outdoor location and orientation of the vehicle. An AWC sensitivity to the wheelchair's location and orientation is significant in terms of safety. The process is responsible for critical feedback to the control unit to dictate kinetic profile. In this thesis, various choices of indoor location determination methods are explored.

2.2 Environment (work-space/task-space) Measurements and Observations

In order to strengthen safety measures to an AWC, accurate assessment of the environment is another critical task. This can be achieved with the fusion of various sensors. There are sensors which can distinguish color and have high resolution but take large computing resources and lack proximity. On the other hand, sensors such as ultrasonic sensors and IR sensors provide accurate and fast proximity calculation but lack resolution and color distinction. This study proposes a method captures proximity and resolution without requiring large computational resources. However, safety can be further enhanced by fusing other sensors.

Table 2.1: Sensor coverage for obstacle assessment

Sensor	Proximity	Resolution	Bandwidth	Color	Object Identification
Depth Camera	Average	good	poor	good	good
RGB Camera	Poor	good	average	good	good
IR Sensor	Good	average	good	poor	poor

Figure 2.1: Sensor coverage for obstacle assessment with or without fusion

2.3 Autonomous Path Determination and Strategies

There are various levels of autonomy but in this research, the development work is done towards the highest level of autonomy that includes adaptive path planning. In particular, the proposed work developed a simulation tool using MATLAB® to test various path planning algorithms under some key list of assumptions.

2.4 Control Unit

This is the central processing or decision-making mechanism where all the feedback and actuating instructions to control motors for locomotion and orientation take place. The control unit is beyond the scope of this thesis but that does not reflect the crucial role it has in order to achieve an AWC.

3. Localization and Orientation

Testing any localization and orientation methods on actual powered wheelchairs requires large resources including but not limited to:

(1) Test environment

(2) Functioning feedback and control system and

(3) Safety measures for the safe operations of the actual wheelchair under testing conditions.

The tests practiced for the development work proposed in this thesis obtained on a small-scale mobile robot that removed any dependency on wait period for the actual powered wheelchair to function at its full capacity. This study presents a typical inexpensive academic robotic competition environment setup with cost effective navigation and mapping solutions for robots in a micro-environment. The outline of the indoor localization study is as follows: At first, a suitable mobile robot hardware is selected with a specific configuration in order to conduct an experimental task. This is followed by presenting three different navigation and mapping techniques. In particular, (1) Odometry, (2) Array of beacons and (3) Computer vision, which are tested, evaluated, and discussed in this thesis. A cost comparison is presented with commercially available alternative solutions.

3.1 Experiment Setup

In this thesis, a mobile robot with fixed wheels is used as the general robotic platform to present and discuss mobile robot localization and mapping techniques. These techniques are tested with a simple experimental task, which combines wireless communication, navigation, and mapping tasks. The zumo robot as shown in Figure 3.2, used as the generic mobile robot platform is powered

by battery at the bottom of the robot's chassis. An Arduino Uno micro-controller module is used to run the respective logic and processing commands. In addition, a Zumo Arduino shield is utilized with the objective to reduce the number of jump wires. This shield has an integrated DRV8835 DC motor driver along with an integrated LSM303D 3-axis accelerometer with a 3-axis magnetometer and an integrated L3DG20H 3-axis gyroscope, that can be used to track acceleration, orientation, and rotation of the robot. The robot is assembled with a 75:1 micro metal geared motor (HP). The robot body frame of reference and fixed world frame of reference is shown in Figure 3.1. The parameter *r* from Figure 3.1. (a) represents the radius of each wheel and the parameter *2d* is the shortest distance apart between two rear wheels. The assembled robot with the Arduino Zumo shield (without Arduino) is shown in Figure 3.2, contains convenient spots for the placement of infra-red sensor array, indicated as "general purpose I/O for sensor modules" in the figure, in order to detect any marked boundaries of the work space. A square plain with a flat surface serves as the main platform with dimensions of 2 meters by 2 meters that includes dark boundary markers for the onboard infra-red sensor arrays to detect the borders of the workspace. The robot task is to move in a straight line until it reaches the border followed by a random amount of rotation in order to pick a random direction. This is repeated after orienting in the new direction until the robot hits the next border marker. The process repeated until some termination event occurs. The robot is equipped with photosensitive resistors to detect any distinct source of light. Once the robot reaches a spot within the platform with greater amount of light projected, the robot task is terminated. A computer is located outside of the workspace with the task to communicate with the robot using xbee modules. The communication entails any local computations needed to be shared. The computer maps the movements of the robot as the robot performs its task. In this work, MATLAB® tools are used to develop any mapping program.

3.2 Odometry Navigation and Mapping

A theoretical model for standard fixed wheel rear drive mobile robot kinematics is needed to develop the relative position measurement or dead reckoning for localization and mapping suitable for a small workspace, [18]. The robot wheels are assumed to have no slippage between the drive surface and the wheel itself. Equations (3.1) and (3.2) describe the Jacobian matrix associated with the wheel speed vector and the inverse Jacobian matrix with the robot's twist vector to produce $\vec{W}$ & $\dot{\vec{\theta}}$ respectively. Where $\dot{\theta}_1$ and $\dot{\theta}_2$ are angular velocities as indicated in Figure 3.1(a). The position of the robot with respect to the world frame are presented using *x* and *y* coordinates. The orientation or rotation angle with respect to the horizontal axis of the world frame is indicated in Figure 3.1(b) using ϕ.

$$\vec{W} = \begin{Bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{Bmatrix} = \begin{bmatrix} \frac{r}{2d} & -\frac{r}{2d} \\ \frac{r}{2}\cos(\phi) & \frac{r}{2}\cos(\phi) \\ \frac{r}{2}\sin(\phi) & \frac{r}{2}\sin(\phi) \end{bmatrix} \begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{Bmatrix} \tag{3.1}$$

$$\dot{\vec{\theta}} = \begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{Bmatrix} = \begin{bmatrix} \frac{d}{r}\cos(\phi) & \frac{1}{r}\cos(\phi) & \frac{1}{r}\sin(\phi) \\ -\frac{d}{r}\cos(\phi) & \frac{1}{r}\cos(\phi) & \frac{1}{r}\sin(\phi) \end{bmatrix} \begin{Bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{Bmatrix} \tag{3.2}$$

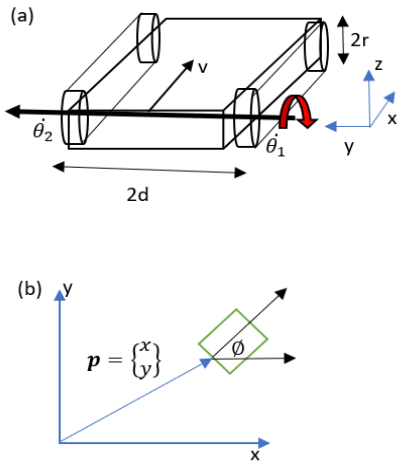


Figure 3.1: (a) Mobile robot in body frame; (b) Mobile robot in fixed frame

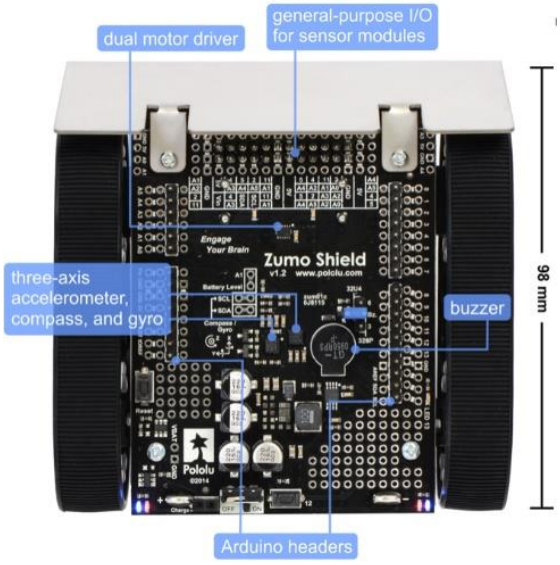


Figure 3.2: Assembled Arduino Zumo shield

Therefore, straight line motion of the robot platform is generated when $\dot{\theta}_1 = \dot{\theta}_2$ while the rotation of the robot about the center point of the axis that connects two rear wheels as shown in Figure 3.1(a) occurs when $\dot{\theta}_1 = -\dot{\theta}_2$. These – straight line and rotation – are the only motions required for the robot task.

The first navigation method for the specified mobile robot platform is based on the integration of small steps of the robot's movement in a 2D workspace. Each step is defined by the wheel speed and step duration. The wheel speed is controlled by a pulse width modulated signal from the micro-controller module that is sent to the motor driver. The parameterization of the navigation process is given as follows: Suppose the time duration of each step is denoted by Δt. The angular velocities of the right and the left wheels are $\dot{\theta}_1$ and $\dot{\theta}_2$, respectively. At any given ϕ, the discrete steps in both coordinates are computed as $\Delta x = \dot{\theta} r cos(\phi)\Delta t$ and $\Delta y = \dot{\theta} r sin(\phi)\Delta t$, where $\dot{\theta} = \dot{\theta}_1 = \dot{\theta}_2$ is used for straight line motion of the robot. Also, the rotational step $\Delta\phi$ is extracted from $\dot{\phi}\,\Delta t$, where $\dot{\phi} = \frac{r}{d}\dot{\theta}$ and $\dot{\theta} = \dot{\theta}_1 = -\dot{\theta}_2$ for clockwise/counter-clockwise rotations. For the aforementioned test robot, an assumption can be made that the wheel speed remains constant throughout the runtime of the robot tasks, and hence, the speed is assumed to be *K*. This constant is incorporated in the Arduino program for the purpose to test the proposed concept. Given the reference coordinate (x_0,y_0) and control input to alter ϕ and $\dot{\theta}$, the local position can be evaluated using simple iterations:

$$x_{n+1} = x_n + \dot{\theta} r cos(\phi)\Delta t \qquad (3.3)$$

$$y_{n+1} = y_n + \dot{\theta} r sin(\phi)\Delta t \qquad (3.4)$$

$$\phi_{n+1} = \phi_n + \frac{r}{d}\dot{\theta}\Delta t \qquad (3.5)$$

This odometry technique comes with the cost of error development due to the integration of small errors over the duration of the robot operation. This error development is due to both systematic and non-systematic errors. Systematic errors generally refer to an error due to inaccurate calculation of the wheel rotation, incorrect readings from the inertial measurement unit

components and other erroneous measurements from on board sensors, all contributing to navigation errors. In order to develop accuracy in localization, odometry is often used along with other proven methods of navigation and mapping, [19]. However, the cost of implementation for this method is significantly lower relative to some of the commercially available solutions. These commercial solutions may cost approximately ten times more than in a class solution, [20]. Commercial solutions often use expensive but highly accurate and self-calibrating sensors along with hardware that may not serve an academic purpose. Mapping software comes with some of the available indoor positioning systems and are often proprietary, expensive and provide little room for alteration and improvements. The above-mentioned techniques and setup use a Zigbee personal area network for transmitting local calculation results for further work in the base system to generate 2D mapping work. Therefore, selecting odometry techniques depends on the level of accuracy required and the allowable tolerance given to the robotic system while meeting robot task/mission assignment objectives.

3.3 Array of Beacons

In the previous section, the local position of a mobile robot is measured using an open-loop scheme. Any amount of slippage or change in the motor speed beyond measurements will generate inaccurate coordinate calculation. Such changes could be a result of decaying batteries, changes in surface conditions, etc., [21]. Another approach to determine the location of a mobile robot in a 2D workspace is to add an external system that is sensitive to the robot motion. In this thesis, only the coordinates of a single mobile robot on a planar platform is considered. The following sections detail two such methods for localization of a mobile robot using an external setup that is sensitive

to the robot's displacement. One approach is to design a radar like system, where a spatial signal (ultrasonic or laser) is transmitted from a reference point to the robot workspace while sweeping across the span of the platform. Orientation of the transmitter and the characteristics of the reflected signal at the receiver from the robot, determines the location of the mobile robot as shown in Figure 3.3(a) where, coordinates are measured in terms of r and ϕ. The limitation to the radar-based localization method using inexpensive "of the shelf" sensors is given by the dispersion of the received signal during rotation of the transceiver. This inaccurately alters time of arrival of the signal received. Hence, there will be an accumulation of errors in the calculation of r. The second approach is to use an array of sensors, where each sensor is a transceiver that determines the distance from the reflected object (robot) on a given workspace. Each sensors are arranged such that collectively all the sensors address discrete points of a vertical axis in a cartesian plane. The position of the respective sensor and the corresponding signal reflected from the tracked robot provide sufficient information about the target robot location. Some major limitations in the array of beacon approach are due to the resolution of the grid points and choice of sensor. The resolution of the inferred localization information can be easily improved by increasing the density of the number of sensors within the array direction. As the number of sensors increases, the cost of the setup increases as well. As this study addresses a workspace that is no more than two meter in dimensions, it is possible to keep the number of sensors at some minimum while achieving a certain level of accuracy.

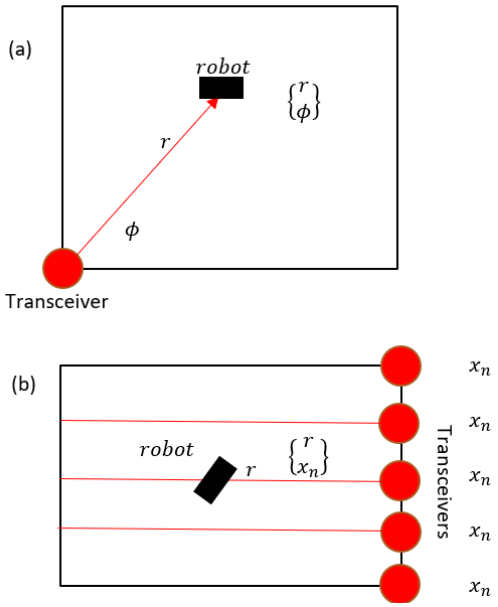


Figure 3.3: (a) Radar based localization; (b) Grid from sensor array

Figure 3.3 shows both configurations discussed in this section. Two types of sensors are used to test these two methods: Ultrasonic sensors are the simplest to implement but produce poor performance in terms of localization accuracy when the resolution of the grid is increased and the robot is the furthest from the sensor array. This is due to the wide arrival angle of the signal. On the other hand, a laser transceiver has a sharp angle of arrival and hence produces an accurate localization. However, the option of an array of lidar increases the cost of implementation. In the following a detail discussion is provided about the choice of inexpensive sensors. For the described experimental robot task, there are several distance measuring sensors available. The choice of the specific sensor for mapping robot locations depends on the desired or required accuracy of the inferred localization. An ideal sensor will transmit the corresponding wave with an associated angle of arrival and departure, as narrow as possible. Otherwise, the sensors will receive reflected signals from a position different from the line of sight set by the sensor orientation. In addition, the measure of the distance can be done in two ways for most distance sensors: the time of flight method is used when the speed of propagation of the transmitted signal is known. The recorded

time lapse for the reflected signal times the speed yields twice the distance travelled by the signal. The second method is to use the signal strength from the reflected signal to calculate the corresponding distance. However, the signal strength has generally a non-linear relationship with the distance and requires some complicated computation in order to extract an accurate result. In this section, two choices of localization processes are discussed. Initially, a radar method is attempted with a laser transceiver. The corresponding sensor for this setup is a VL53L0X. This particular sensor has a sharp beam and a multipath error coming from the received signal when the robot is in movement. The sensor is placed on top of a servo motor that allows for a sweeping motion. The corresponding results are inconsistent and produce dispersions due to the sweeping motion. The sensor grid approach is tested as well using the same laser transceiver as described above. The VL53L0X sensor communicates through an I2C protocol and has the same address value as any other VL53L0X unit. Hence, a laser sensor array – employing a number of these sensors - can only be used if the I2C mux is installed in the system in order to read and distinguish each sensor. Adding I2C mux or an array of ultrasonic sensors will introduce additional challenges as time sharing schemes are required to read from all the sensors. The ultrasonic sensor array is tested with equal distances between each sensor. The distances between each sensor have to be selected carefully with respect to the overall distances. This is because the overlap in angle of arrival/departure cone of signals from different sensors increases with the larger range. Once the localization information is generated by the external setup, the position information is passed to the base system for mapping and visualization using tools from MATLAB®. Considering all the limitations addressed in this section, the following section provides another approach that yields greater accuracy and lesser hardware setup with a cost of more computation effort by the base system.

3.4 Computer Vision Based Positioning

In general, the number of computations per seconds among workstations has been growing rapidly for last few decades. This means that more computationally rich tasks are possible without an additional cost. MATLAB®'s image processing toolbox comprises a large set of post image processing tools.

Fundamental computer vision algorithms along with the image processing toolbox added new solutions to the search for cost-effective mobile robot localization problem for small-scale platforms. Figure 3.4 shows a generic approach for identifying a target object location from an image. At first, a few assumptions need to be made prior to the discussion regarding the experiment conducted. The position and projection of the camera is kept at the center of the platform with fixed height and a vertically downward direction, respectively. Given the dimension of the platform, and assuming shape and size of the target mobile robot remains unchanged, the following steps are applied to track the mobile robot. The entire process can be divided into two major sessions. The first session is composed of the initialization step followed by an iteration step. In the initialization step, the camera object is initiated to begin the recording process and an image with the target robot is captured. The dimensions of the captured image is shrunk from a 3D to 2D version by converting the acquired RGB image to grayscale. The process of edge profile using 'Sobel' method is generated next. In the proposed process, the user is asked to crop manually a tight segment of the edge profile from the image that covers the target robot. The cropped segment is used as a convolution filter image to the future stream of images once the address map system starts running. Once the system run status is enabled, the iteration step process starts. A portion of the iteration step process is identical to the initialization process. A loop of jobs is considered that includes image capture, RGB to grayscale conversion, derived edge profile processing, followed

by the convolution filter processing using the cropped reference from the initialization step. A map of matching indexes is generated where the maximum indices indicates the greatest probability of having the tracked robot being in the corresponding position. The iteration step is repeated as long as the tracking status is enabled.

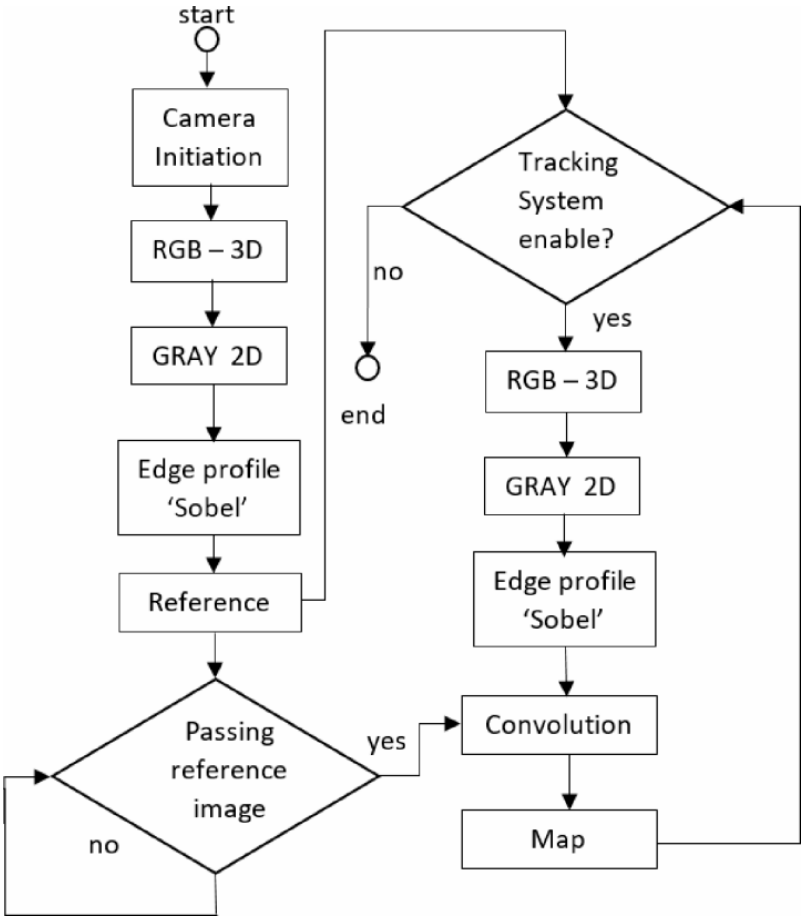
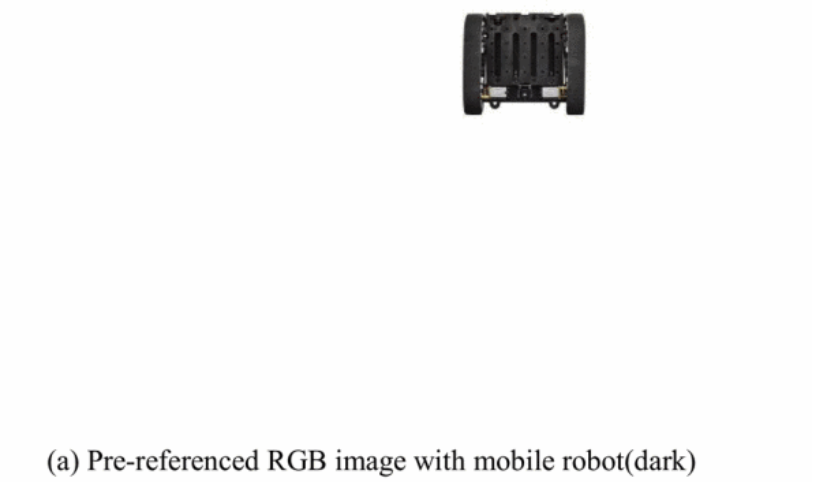


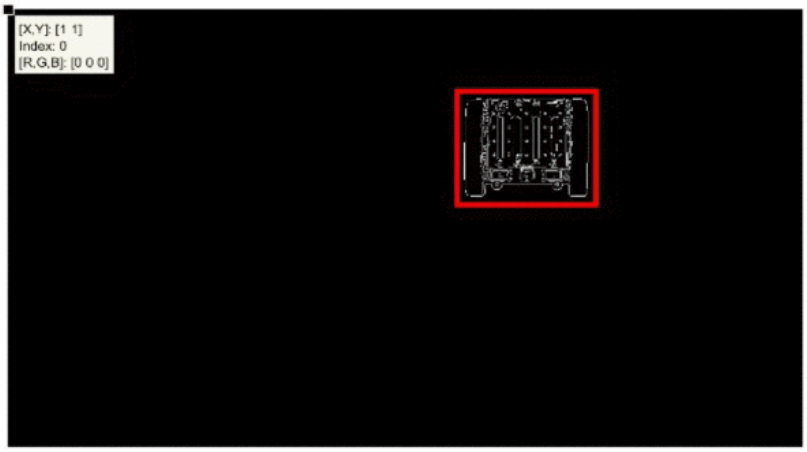
Figure 3.4: Positioning using computer vision algorithm flowchart

Figure 3.5(a) shows pre-reference RGB image prior to any post processing. Figure 3.5(b) shows the cropped reference image with edge profile process after the RGB to gray conversion is performed. Once the tracking process is initiated, a stream of plots produced from the instantaneous localization is created in order to generate a real-time visual feed of the robot's

position. Figure 3.7(a) shows the localization as a surface map after each convolution step is performed. Following Figure 3.7(b) shows a contour map corresponding to the surface plot.



(a) Pre-referenced RGB image with mobile robot(dark)



(b) Cropping segment from edge profile

Figure 3.5: Image process for position determination

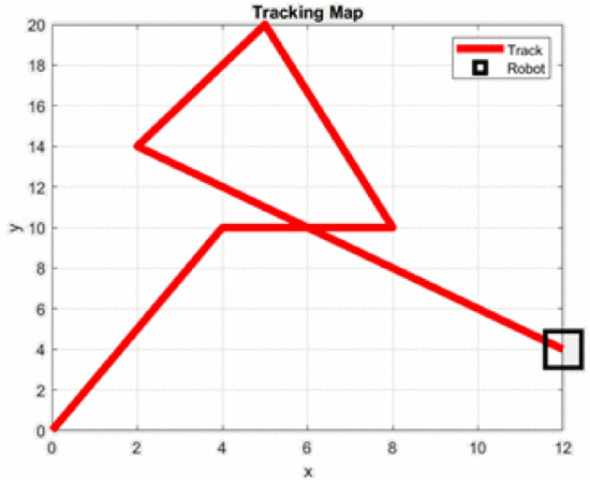
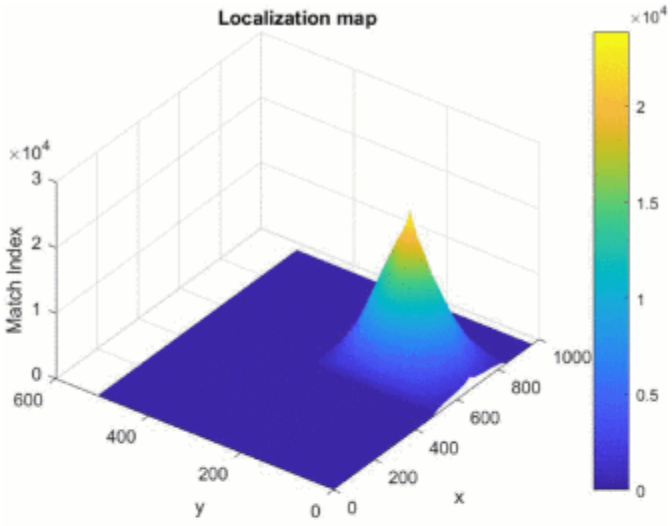
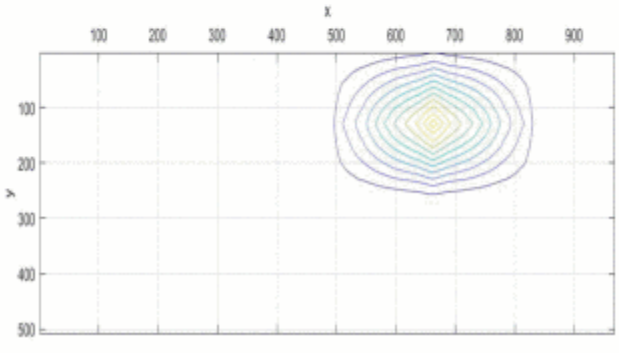


Figure 3.6: 2D workspace mapping on single mobile robot



(a) Position map (surface)



(b) Position map (contour)

Figure 3.7: Snapshots of real-time visual position feed

The output of the computer vision approach shows perfect accuracy for the experimental task described and corresponding mobile robot hardware used. There is some amount of lagging due to the visual feed from the base computer. However, the key advantage here is that the mobile robot is free from doing any position calculations and computations. Instead, the robot communicates with the base system with a predefined sampling rate to receive its current location. The computer vision setup is relatively more expensive than other solutions discussed in this thesis. Two key components of the expenses are the quality of the camera used and the capacity of the base system. However, the camera and the workstation both have a wide range of costs associated and often academic laboratories stock such components. Some of the commercially available software/hardware pair choices for computer vision-based positioning system require a high premium for the use of proprietary software along with corresponding hardware. MATLAB® is also a cost factor but often available under college site-licenses. However, software tools used for the computer vision-based positioning solutions can be easily achieved using open-source tools like Python and Octave.

3.5 Localization Performance Analysis

The three approaches discussed in this thesis are summarized in Table 3.1 with respect to components and technique used, as well as their disadvantages and advantages. Odometry has gained tremendous popularity in position determination of a mobile robot for academic assignments. This technique works even better when the operation time and dimension of the robot workspace is small. This is mainly due to fewer opportunities for accumulation of the error. Additionally, it is also independent of the number of robots present in the workspace. However,

as the operation time increases or the robot receives any unexpected impact or event that triggers a large error, there must exist an alternative method to reset the position information through the help of an external system. An array of beacons could represent such a solution. In this research, a simple experiment is used as a setup for a single mobile robot in a predefined workspace. If a high resolution of the robot's position is not required, an array of beacons approach could be a fast solution to position determination. If one is looking for a localization solution that has great accuracy (10 mm or less) along with independence from the number of mobile robots participating in the task space, the computer vision-based (CV) solution would be an appropriate choice. The CV approach expenses may vary based on choice of hardware. However, for the experiment performed in this thesis a low-resolution web camera is sufficient.

Table 3.1: Comparison among indoor localization approaches offered in this thesis

Approach	**Odometry**	**Array of beacons**		**Computer Vision**
Technique	Local positioning computation by tracking micro-movements	Rotating transceiver (rotating radar)	Linearly arranged array of transceiver	Capture image of target object to generate edge profile followed by cross matching with reference.
Sensor(s)	Accelerometer, gyro, magnetometer, motor rotation decoder	Ultrasonic sensor or LIDAR		RGB Camera
Advantage	Relatively inexpensive and simple to implement.	Relatively simple to implement and faster to calculate position coordinates.		Highly accurate for small platform addressed in this thesis.
Disadvantage	Prone to systematic and non-systematic error (e.g. slippage, surface).	Multipath error, resolution of position grid proportional to number of sensors used.		Computationally rich and choice of hardware may be expensive.

In this chapter, regardless of the approach taken, there is an additional mapping task performed in the base system using tools from MATLAB®. Similar mapping tasks can also be achieved using opensource tools such as Python or Octave, as mentioned earlier. Figure 3.6 shows the mapping results of a single robot on the 2D platform. There are exciting open-source tools available such as ORB-SLAM or ROS navigation to compute simultaneous localization and mapping (SLAM). However, this thesis is tailored towards educational needs for students from a wide range of backgrounds and not limited to robotics stream. The thesis further emphasizes the use of inexpensive hardware with minimal software complexity. The objective in this work aligned with addressing the localization issues associated with the autonomous wheelchair development. However, the goal is also to include an inexpensive solution so the research can be conducted using common resources available in most institutions. Most institutions apply MATLAB® in various other fields due to its rich set of tools that allow students to explore and mesh concepts in an interdisciplinary fashion, without having additional complexity to implement. MATLAB® tools and hardware used in this study are aimed for a known small-scale environment and single robot with opportunity to include machine learning and deep learning processes with ease. The problem can be extended to multi-robot and unknown environments where relevance of using tools such as ORB-SLAM or ROS navigation is more significant, [22].

4. Environment Measurements and Observations

This chapter discusses two of the four streams associated with the development of the AWC. (1) The feedback system and (2) the navigation strategy including mapping and simulation. The work emphasized minimizing computational load by selecting resource constraint hardware such as single-board computers and other low-power co-processors to achieve a critical aspect of the feedback system. The feedback system proposed in this chapter utilizes a pretrained convolutional neural network to process captured images from the environment. The object detection process using cameras has been around for awhile, [23]. However, in this work we propose the use of depth image merged with convolutional neural network-based object detection transfer learning to identify obstacle classes and their proximity. Moreover, the tools and the development work associated with the proposed feedback system are open source. Furthermore, the work proposes low-cost navigation, mapping, and simulation processes suitable for extending AWC development. The navigation, mapping, and simulation tasks are developed using a low-cost research software tool such as MATLAB® and Simulink.

The following items are included in the Appendix for detail setup resources:

- Platform configuration
- RGB + Depth Image processing using tools
- Co-processor: Google Coral

4.1 Hardware and Software Configuration

The feedback system is comprised of multiple types of sensors. In this work, the focus is partly on computer vision (CV) applied to the AWC. The CV pipeline's hardware and software tools are discussed in the following section.

The CV setup is constructed using a depth camera stemming from the Microsoft XBOX 360 Kinect setup, a single board computer or Raspberry Pi 4 Model B, and an edge TPU co-processor from Google Coral. The depth camera is used to read both the RGB and the depth images from the environment. The images are processed and hosted in the Raspberry Pi (single board computer or SBC) and further relayed to the co-processor for machine learning inference computations. The Kinect camera can capture images using two cameras fixed in positions to enable the trilateration method [24] for deriving depth information. The Kinect has a viewing angle of 43^0 vertical and 57^0 horizontal and allows tilting that ranges $\pm 27^0$. The maximum frame rate offered by the camera is 30 frames per second which is sufficient for the work presented in this thesis. The Raspberry Pi used in this study uses Raspbian OS, 64-bit architecture, 1.5 GHz quad-core processor with 4GB RAM. In addition, the edge TPU from Google Coral is added to focus on machine learning routines, and can perform four trillion operations (tera-operations) per second (TOPS), using 0.5 watts for each TOPS (2 TOPS per watt). The utilized wheelchair is a power chair (PermobilTM M300 Corpus HD) with joystick navigation control, providing power with the use of two 12V batteries. Note, none of the OEM control systems were altered for the proposed work. Instead, a mechanical connection to the joystick is envisioned to connect the control commands from the proposed control system to the wheelchair. In this fashion, the type and manufacture of the wheelchair is irrelevant as any such device can easily be modified.

Specific software tools are selected to enable the streaming process from the camera to the Raspberry Pi. The freenect tool is used to verify the functions of the depth camera. Additional dependency files such as libgl-mesa-swx11 and equivalent are required for OpenGL utilization. The RGB image/object classification routine with the pre-trained model is used for object detection. The routine is written in Python and requires OpenCV along with the libraries from python-freenect supporting the depth camera. Moreover, TensorFlow lite is applied to accommodate computationally minimized versions of the object detection processes. The processes are executed at the co-processor hardware. Additional tools are used to support the co-processor hardware such as libedgetpu1-std. Figure 4.1 shows the hardware in sequence to support the workflow proposed in this thesis.

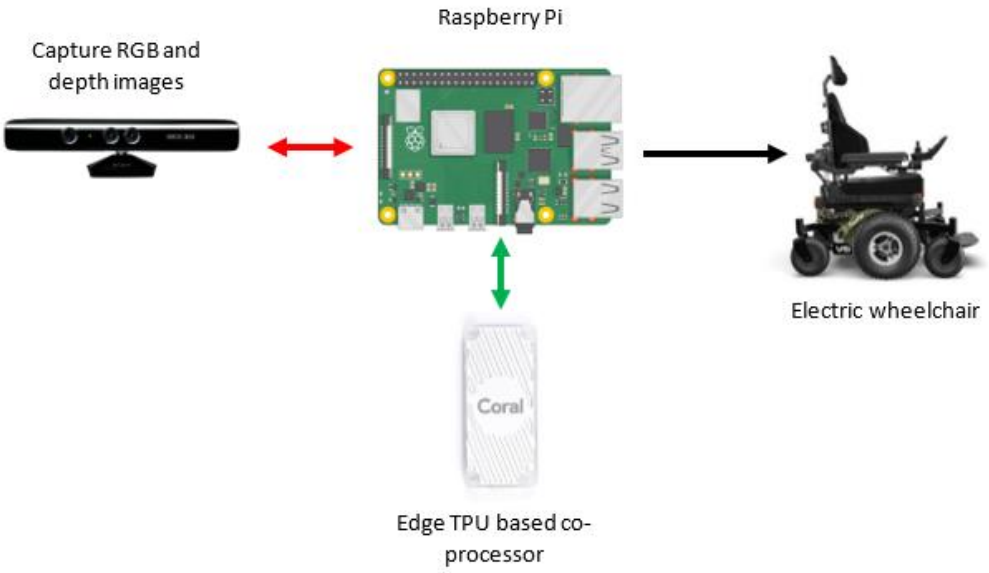


Figure 4.1: Workflow of feedback system proposed in this thesis

4.2 Computer Vision for Obstacle Avoidance

The computer vision process starts with capturing images using both cameras available in the Kinect. The depth images are constructed using the RGB images from the two cameras separated by a fixed distance using the trilateration method. A routine is developed to read both the RGB images and constructed depth images but feeds only the RGB images to a pre-trained convolutional neural network (CNN). The network is TensorFlow Lite-supported object detection routine in real-time. The network or the object classifier outputs the detected object label, the probability of the detected object, and the detected object's boundary. The depth image color scheme is changed to a single channel with values ranging from 0 to 255 where 0 (light pixels) indicates furthest or undetermined objects and 255 (dark pixels) indicates the nearest object. The object boundaries from the object classifier are mapped on the depth image. The pixels from the mapped area of the depth image are used to calculate the mean/max channel values. The mean/max channel values within the detected object's boundary of the depth image addressed by the RGB object detection network are strongly correlated with the object's proximity. Hence, from the process described above, there are two quantifiable outputs generated at every frame: (1) The probability of the detected object (p) and (2) the mean/max channel values from the depth image correspond to the detected object location (d). In addition, the quantity x can be derived from p and d by using a linear combination as shown in Equation 4.1, where a and b are constant coefficients used to scale the detected obstacle's probability and the obstacle's proximity respectively. The x term can be used towards the obstacle avoidance process to determine the obstacle's proximity and severity. Optimization of Equation 4.1's coefficients are beyond the scope of this work. Also, the arrangements of p and d to form Equation 1 are not explored in this study for best results.

$$x = ap + bd \qquad (4.1)$$

The objective of the CV process is to detect obstacles by the AWC and make autonomous decisions to avoid any harmful contacts. Therefore, the process needs to be executed in real-time with minimum delay. The process is achieved at about 12 frames per second - at most - after including a co-processor, specifically the Google Coral (edge TPU). The inference steps are computed at the co-processor with enhanced performance relative to the Raspberry Pi. The identical routine for object detection runs at 1-3 frames per second without the edge TPU (also called accelerator). The CV process discussed in this section is neither the complete feedback system for the AWC nor is it mounted on the chair in full operation mode. To enhance the safety features of the AWC, there should be additional proximity sensors and methods in place such as ultrasonic, infrared, or laser-based obstacle and edge detection methods. Figure 4.2 – 4.4 show forms of images applied in the obstacle avoidance process. Figure 4.2 shows the RGB image for the object detection process that eventually merged with the mono channel depth image shown in Figure 4.4. The detected object label with the probability in black text and area in the green rectangle indicating pixels for proximity calculation is shown in Figure 4.4.

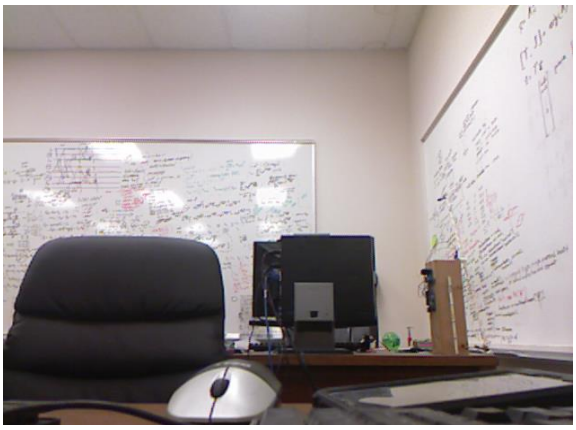


Figure 4.2: RGB image captured from the Kinect before feeding to the object detection process

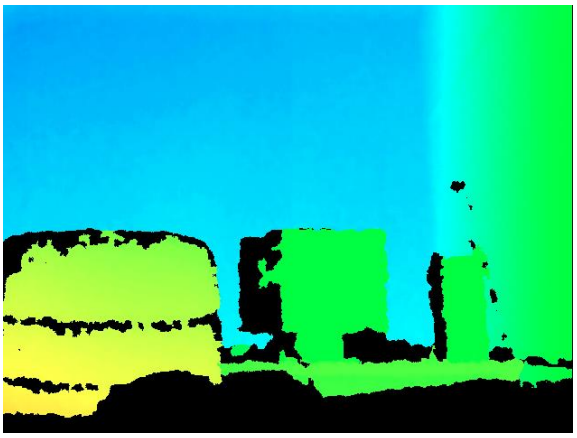


Figure 4.3: Depth image captured from the Kinect

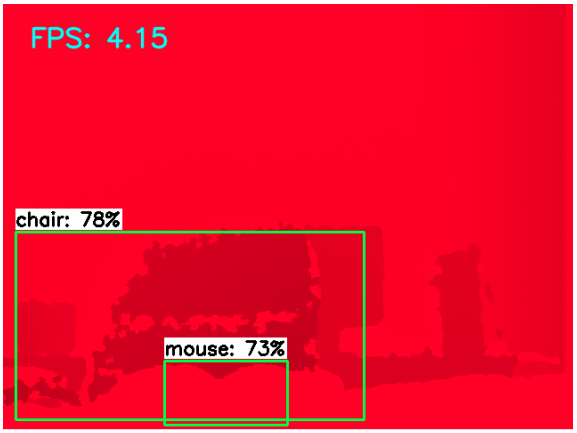


Figure 4.4: Depth image to mono channel followed by detected object's boundary mapping for proximity calculation

4.3 Navigation Strategies and Simulation

There are four key areas of research that guide AWC development as mentioned in the introduction. In general, it is challenging to conduct a study on navigation and mapping strategies with the physical system during the development phase of the other key areas. Hence, the simulation path is considered to remove dependencies on the control system, the localization/positioning, and the feedback system. There are key assumptions considered during the simulation and mapping computations which are listed in the followings:

- Localization or position of the AWC is known and accurate.
- Obstacle detection processes are fully functioning and accurate.
- The AWC is running on a flat surface.
- Direction and velocity control are functioning and accurate.

The simulation is designed using MATLAB® basic toolkit. The aim of the simulation is to keep the navigation algorithm expandable with minimal effort. The resources used in the simulation design are found with low cost to no cost depending on the purpose of the use. Hence, the simulator is suitable for academic use. Based on the assumptions made above, the simulation is based visually on a grid covering the entire workspace of the AWC, which is shown in Figure 4.5. Each of the tiles represents possible coordinates that the AWC is capable of reaching. However, some tiles are marked with filled colors to indicate obstacles, the AWC current position, and destination. Figure 4.5 also shows the trace of the journey made by the AWC during the simulation. As the AWC travels through each block, an iteration count is placed to observe the overall cost of the journey. The navigation procedure considered and as indicated in the pseudo-code and the flowchart shown in Figure 4.6, minimizes the iteration cost in most scenarios. Each scenario is

created with randomly placed obstacles, a random starting point, and random destination coordinates. Once the simulation starts to iterate, each incoming step for the AWC is calculated. The next step of the AWC is calculated based on the current position and surrounding obstacles. The priorities considered for the future step calculations are addressed in the ***next step process*** described in the pseudo-code below. The flowchart in Figure 4.6 shows the overall route determination processes applied in the simulation.

Pseudo-Code

Motion list: {right, left, forward, and backward}
While *all the neighbor tiles are not evaluated*

    Pick a tile from the possible motions list that
    has not been picked in the current loop

    If *the tile picked has an obstacle*
        Discard the motion choice
    Else
        Store the motion choice
        to the next step vector.
    End If
End While

If *the next step vector is empty*
    The next step is the previous position.
Else
    Euclidean distance calculated between
    each of the elements available in the
    next step vector and the destination.

    The next step vector is sorted from
    the least to the greatest distance.

    Each coordinate found in the next step
    choices are re-arranged based on past visits.

    The next step is the least Euclidean distance
    coordinate among the least visited options.
End If

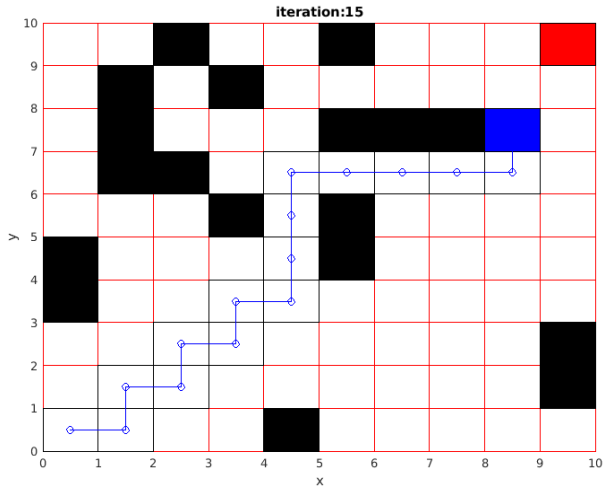


Figure 4.5: Simulation grid showing the AWC tile as blue, destination tile as red, obstacle tiles as black along with a trace of AWC journey using blue line.

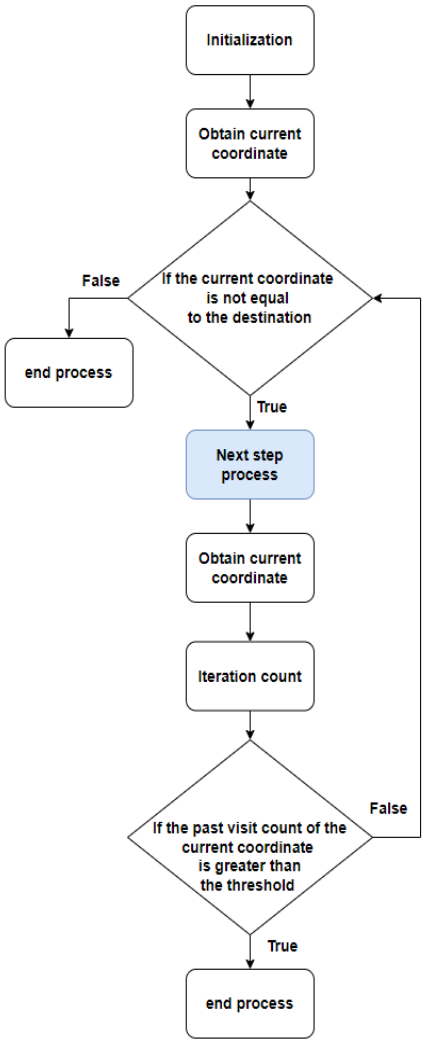


Figure 4.6: The overall route determination processes applied in the simulation

4.4 Conclusion

Navigation of AWC's may fill a need in hospital logistics, aging in place, as well as in support and enhancement for individuals living with a disability. The proposed autonomous functionality employs an existing power wheelchair, with an add-on that connects to the existing wheelchair control joystick, using mechanical means. In addition, readily available off-the shelf components are proposed and tested for constructing the vision-based navigation system, such as the Kinect camera system, a single board computer such as the Raspberry Pi 4, and the Edge TPU co-processor along with freely available software such as OpenCV, python-freenect, and TensorFlow, etc. This proposed hardware concept is engaged by a pre-trained CNN, allowing for obstacle recognition and hence avoidance during autonomous driving of the AWC. In addition, we also propose a novel integration of the two camera feeds for discriminating against obstacles of less importance due to lack of proximity. A proposed navigation algorithm is tested using simulations, with the objective to test if the overall concept is feasible for sensing, controlling, and making decisions during the autonomous driving portion of an AWC. The proposed concept and algorithms as well as hardware and software implementations are currently not optimized, which is part of the future work on this project. The presented simulation work and testing of the proposed obstacle recognition system indicate the proposed hardware and software concepts are feasible for further development and integration into a power wheelchair, provided additional sensors are embedded to provide for safe operation of the AWC.

5. Additional Studies

5.1 Vicon – Motion Capture

5.1.1 Experiment Setup and Procedure

In order to address quality of the motion captured data, a simple experiment is performed. The Vicon system is prepared by making appropriate calibration, masking and capturing a test motion using the calibration wand as a subject. The calibration wand contains five markers where Euclidean distance between the markers remains constant as the subject is just a T-Shaped metal body with fixed marker position. After capture is completed, data are exported to csv format for further studies under the rich tools of MATLAB®. Once the data are imported to MATLAB®, each marker is plotted in multiple way to aid understanding of the motion capture performance. The followings are the observations considered:

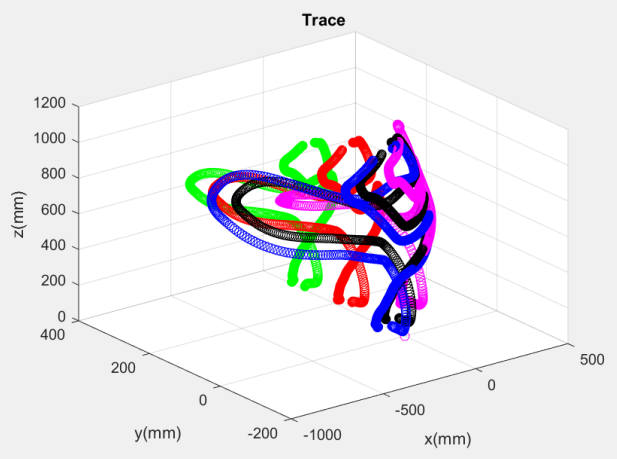


Figure 5.1: Trace of motion from five markers attached to the calibration wand

The purpose of this Figure 5.1 is to observe any ghost behavior during the animation of the captured data. Any sudden jump in data location indicates corrupted data region and hence, should be avoided during performance analysis.

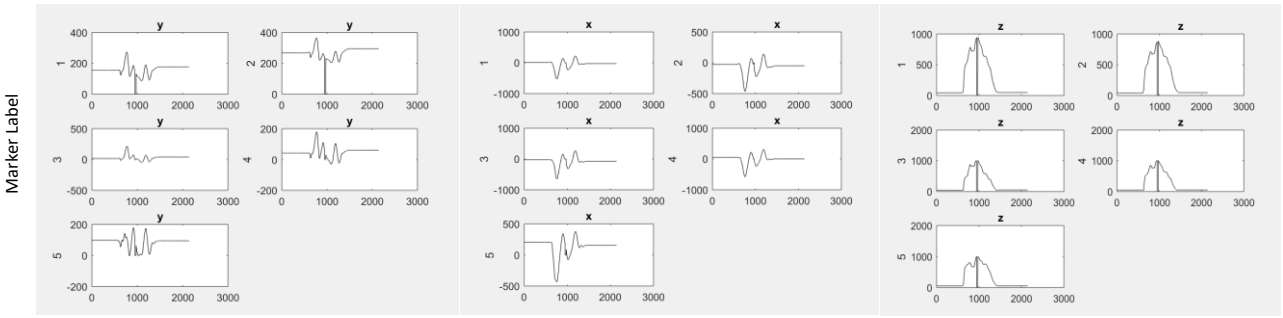


Figure 5.2: Plot of individual marker correspond to each axis

Purpose of this Figure 5.2 is to further identify any corrupt zones where data is missing and observe each marker transition during the capture. Unlike placing the markers on a human subject, this experiment does not contain any human error. Therefore, all error sourced is from either calibration or post process of raw data.

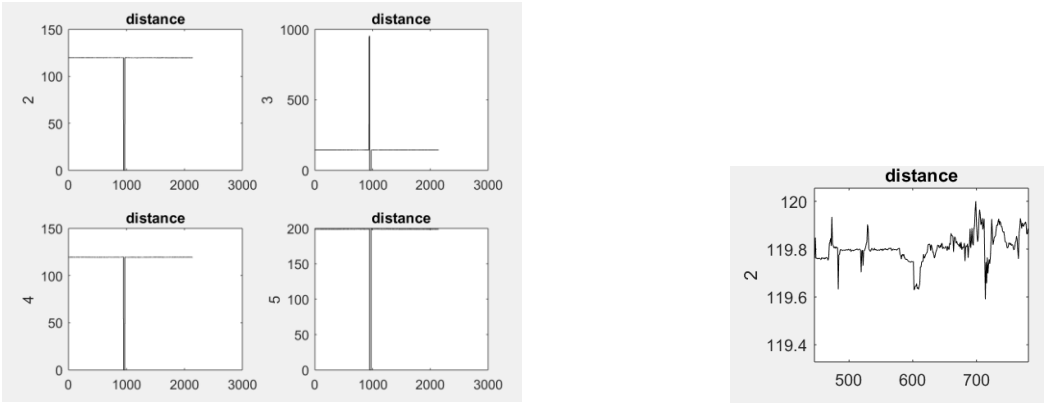


Figure 5.3: Euclidean distances between the markers during motion capture

Ideally the Euclidean distance is supposed to be constant over the capturing period as the calibration wand remained unaltered during the experiment. However, the absolute distance of all markers relative to a single marker (marker 1 in above plot) shows a distortion noisy feature. A broken plot indicates missing data zones. However, the rest of the plot is seemingly a straight line which is expected but unlike an ideal case; some deviation in absolute distance has been observed. The standard deviation and error readings are considered. Refer to the MATLAB® file for the Vicon for greater details (appendix 7.5).

5.2 ONSEMI

5.2.1 Problem Statement

The assigned problem is associated with wafers that get damaged during the handling process. The misalignment issue appears if the wafers are not correctly placed into the carrying unit. The misalignment issue causes damage or breaks to the wafer plates.

5.2.2 The Proposed Work

To mitigate the mishandling of the wafer by the particular automation, the first step is to notify the robot that the wafers are in a misaligned position. Hence, it is important to monitor and watch the positions carefully before the entire process. The proposed solution for this malfunction is, instead of using human eyes use the computer vision and a more specifically an image processing system. To work with the solution, a prototype is taken, where there is a tray which can carry wafers and a camera is mounted in front of the tray which can extract images of the boundary of the wafers, by capturing the front view. To implement this system, MATLAB® is used with an edge detection algorithm from its image processing toolbox. This procedure evaluates the position or alignment of the wafers and generates a flag before any malfunction takes place. The setup is shown in Figure 5.4.

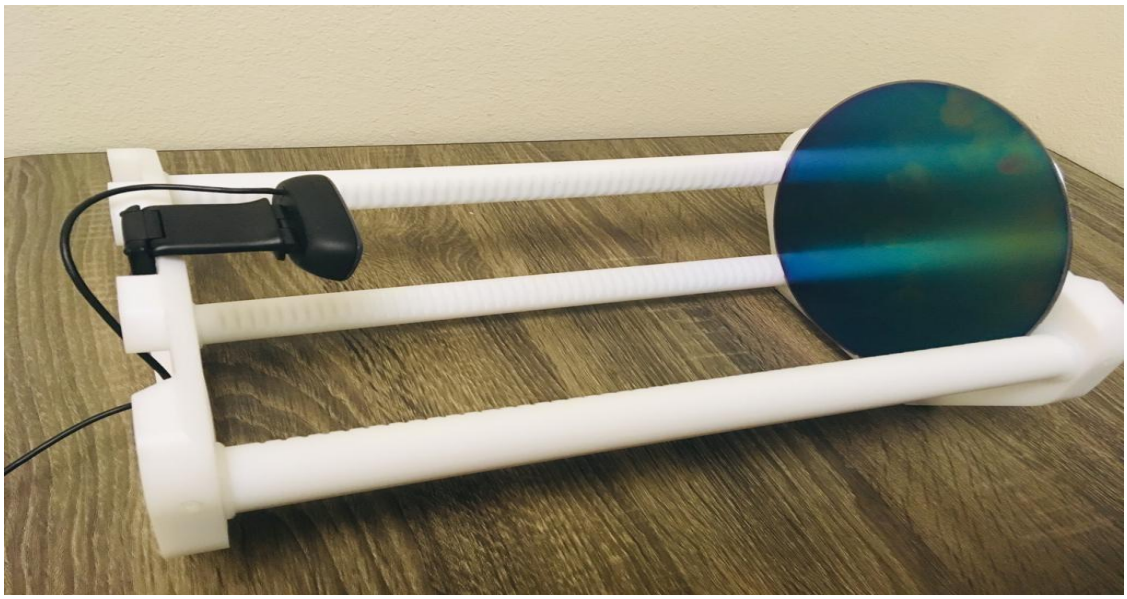


Figure 5.4: Prototype of wafer carrying unit with placing a camera

5.2.3 Technical Description

The following steps have been applied to implement the prototype method: -

- Edge detection algorithm applied to distinct boundaries of the target object. Two images are captured to test the procedure. The first image represents a reference image and is considered to be the ideal orientation of the target object. The image is further processed including conversion to gray scale followed by an edge detection profile generated using the ‘Sobel’ or ‘Canny’ algorithm.

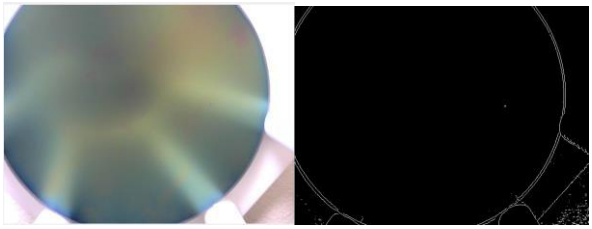
- BW = edge detect edges using the Sobel method. This method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I (image) is maximum.

- Scalar value that specifies the standard deviation (σ) of the Gaussian filter. The default is sqrt (2). Edge chooses the size of the filter automatically, based on σ.

- Evaluation of images are done using a correlation coefficient matrix between the reference and test image. Also squared error sum is applied to aid further distinguish the two images.

- Performance can be enhanced by applying a high resolution camera and mounting the camera on the optimal position (optimal angle of view). For this prototype testing, a "*Logitech"* webcam camera was used, which is MATLAB® supported and its resolution is about *720 pixels*. The higher the resolution, the greater details can be extracted from images. Future research can be aimed at arranging a consistent setup with a more complicated scenario to the tailor image processing further to generate more accurate results.

Reference image (RGB) Edge detection profile (Reference)



Test image (RGB) Edge detection profile (Test)

Figure 5.5: Set of images to determine any misalignment between reference and test image.

6. References

[1]. H. Grewal, A. Matthews, R. Tea and K. George, "LIDAR-based autonomous wheelchair," *2017 IEEE Sensors Applications Symposium (SAS)*, 2017, pp. 1-6, doi: 10.1109/SAS.2017.7894082.

[2]. J. S. Nguyen, S. W. Su and H. T. Nguyen, "Experimental Study on a Smart Wheelchair System Using a Combination of Stereoscopic and Spherical Vision", *35th Annual InternationalConference of the IEEE EMBS*, 3–7 July, 2013.

[3]. R. Alkhatib, A. Swaidan, J. Marzouk, M. Sabbah, S. Berjaoui and M. O.Diab, "Smart Autonomous Wheelchair," *2019 3rd International Conference on Bio-engineering for Smart Technologies (BioSMART)*, 2019, pp. 1-5, doi: 10.1109/BIOSMART.2019.8734264.

[4]. Branco, S.; Ferreira, A.G.; Cabral, J. Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey. *Electronics* 2019, *8*, 1289. https://doi.org/10.3390/electronics8111289

[5]. Ryu, H.-Y.; Kwon, J.-S.; Lim, J.-H.; Kim, A.-H.; Baek, S.-J.; Kim, J.-W. Development of an Autonomous Driving Smart Wheelchair for the Physically Weak. *Appl. Sci.* 2022, *12*, 377. https://doi.org/10.3390/app12010377

[6]. Gao, C., Sands, M., Spletzer, J.R. (2010). Towards Autonomous Wheelchair Systems in Urban Environments. In: Howard, A., Iagnemma, K., Kelly, A. (eds) Field and Service Robotics. Springer Tracts in Advanced Robotics, vol 62. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-13408-1_2

[7]. Royo, S.; Ballesta-Garcia, M. An Overview of Lidar Imaging Systems for Autonomous Vehicles. *Appl. Sci.* 2019, *9*, 4093. https://doi.org/10.3390/app9194093

[8]. Beattie, P.D., Bishop, J.M. Self-Localisation in the 'Senario' Autonomous Wheelchair. *Journal of Intelligent and Robotic Systems* 22, 255–267 (1998). https://doi.org/10.1023/A:1008033229660

[9]. Lankenau, A., Röfer, T., Krieg-Brückner, B. (2003). Self-localization in Large-Scale Environments for the Bremen Autonomous Wheelchair. In: Freksa, C., Brauer, W., Habel, C., Wender, K.F. (eds) Spatial Cognition III. Spatial Cognition 2002. Lecture Notes in Computer Science, vol 2685. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45004-1_3

[10]. C. Wang *et al*., "Autonomous mobile robot navigation in uneven and unstructured indoor environments," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 109-116, doi: 10.1109/IROS.2017.8202145.

[11]. Smids, J., Nyholm, S. & Berkers, H. Robots in the Workplace: a Threat to—or Opportunity for—Meaningful Work?. *Philos. Technol.* 33, 503–522 (2020). https://doi.org/10.1007/s13347-019-00377-4

[12]. Hvilshøj, M., Bøgh, S., Skov Nielsen, O. and Madsen, O. (2012), "Autonomous industrial mobile manipulation (AIMM): past, present and future", *Industrial Robot*, Vol. 39 No. 2, pp. 120-135. https://doi.org/10.1108/01439911211201582

[13]. G. G. Jaman and S. C. Chiu, "A Mobile Wireless Sensor Network Emphasizing Region of Interest via a More Effective Swarm Intelligence Method," 2014 National Wireless Research Collaboration Symposium, 2014, pp. 74-78, doi: 10.1109/NWRCS.2014.18.

[14]. G. G. Jaman, N. Farheen and M. P. Schoen, "Cost Effective Mobile Robots Navigation and Mapping System for Education," 2020 Intermountain Engineering, Technology and Computing (IETC), 2020, pp. 1-6, doi: 10.1109/IETC47856.2020.9249154.

[15]. Watanabe, K., Shiraishi, Y., Tzafestas, S. G., Tang, J., & Fukuda, T. (1998). Feedback control of an omnidirectional autonomous platform for mobile service robots. *Journal of Intelligent and Robotic Systems*, *22*(3), 315-330.

[16]. Iqbal, J., Xu, R., Sun, S., & Li, C. (2020). Simulation of an autonomous mobile robot for LiDAR-based in-field phenotyping and Navigation. *Robotics*, *9*(2), 46.

[17]. Seredyński, D., Stefańczyk, M., Banachowicz, K., Świstak, B., Kutia, V., & Winiarski, T. (2016, August). Control system design procedure of a mobile robot with various modes of locomotion. In *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)* (pp. 490-495). IEEE.

[18]. Joong Hyup Ko, Seung Do Kim and Myung Jin Chung, "A method of indoor mobile robot navigation using acoustic landmarks," Proceedings of IEEE International Conference on Robotics and Automation, Minneapolis, MN, USA, 1996, pp. 1726-1731 vol.2.

[19]. S. Ma, Y. Zhang, Y. Xu, B. Wang, J. Cheng and Q. Zhao, "Indoor robot navigation by coupling IMU, UWB, and encode," *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, Chengdu, 2016, pp. 429-432.

[20]. J. Černohorský and M. Novák, "Mobile robot indoor navigation," 2016 17th International Carpathian Control Conference (ICCC), Tatranska Lomnica, 2016, pp. 151-155.

[21]. L. Delahoche, C. Pegard, E. M. Mouaddib and P. Vasseur, "Incremental map building for mobile robot navigation in an indoor environment," Proceedings. 1998 IEEE

International Conference on Robotics and Automation (Cat. No.98CH36146), Leuven, Belgium, 1998, pp. 2560-2565 vol.3.

[22]. H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99-110, June 2006.

[23]. Malagon-Borja, L., & Fuentes, O. (2009). Object detection using image reconstruction with PCA. *Image and Vision Computing*, *27*(1-2), 2-9.

[24]. Vinh, T. Q., & Tri, N. T. (2015, September). Hand gesture recognition based on depth image using kinect sensor. In *2015 2nd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS)* (pp. 34-39). IEEE.

7. Appendix

7.1 Localization using Computer Vision (.m files)

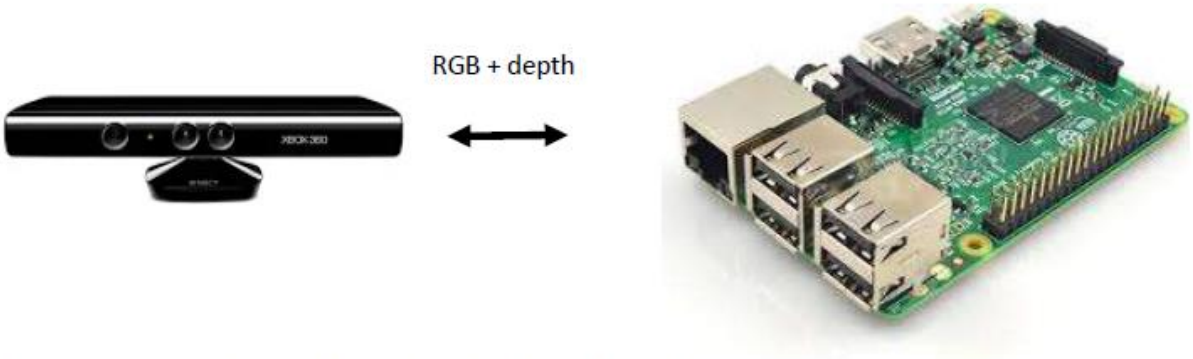
```
%% Image Based Positioning System
% This program intends to produce position of a target object on a 2D plane
% using usb/webcam and convolutional image feature.

%%
% To list the devices and capture image

%%
clear all; close all; clc;
webcamlist
cam=webcam('Integrated Webcam')
%cam=webcam('Logitech HD Webcam C270')
cam.Resolution = '640x480';
preview(cam)

fprintf('Press any key to capture RGB:\n')
pause()
rgb = snapshot(cam);
figure()
imshow(rgb);
closePreview(cam)
%imwrite(rgb,'rgb.tif')
%rgb = imread('circuit.tif');

%%
% Edge detection applied:

%%
gray=rgb2gray(rgb);
edge1 = edge(gray,'Canny');
edge2 = edge(gray,'Sobel');
figure()
imshow(edge1)
figure()
imshow(edge2)

%%
% Define filter

%%
fprintf('Press any key to crop:\n')
pause()
[crop, rect]=imcrop(edge2);
rect=round(rect);

fprintf('Press any key to run mapping:\n')
pause()
map=con2D(edge2,crop);
%x=[start:step:end]; y=[start:step:end]; [X,Y]=meshgrid(x,y)

```

```
figure()
s=surf(map);
colorbar
s.EdgeColor = 'none';

[cmax, iarr]=max(map);
[tmax,j]=max(cmax);
i=iarr(j);

figure
imshow(edge2)
hold on;
rectangle('position', [j,i,rect(3),rect(4)], 'LineStyle','-', 'LineWidth',4, 'EdgeColor','r')
```

```
% 2D convolutional image feature
function[map]=con2D(e,c)

dim=size(c);
w=dim(2);
h=dim(1);

DIM=size(e);
W=DIM(2);
H=DIM(1);

map=zeros(H-h+1, W-w+1);

for i=1:H-h+1
    for j=1:W-w+1
        [f,temp]=imcrop(e, [j,i,w-1,h-1]);
        map(i,j)=sum(sum(f.*c));
    end
end
```

7.2 Depth Image + Raspberry Pi Setup Resources

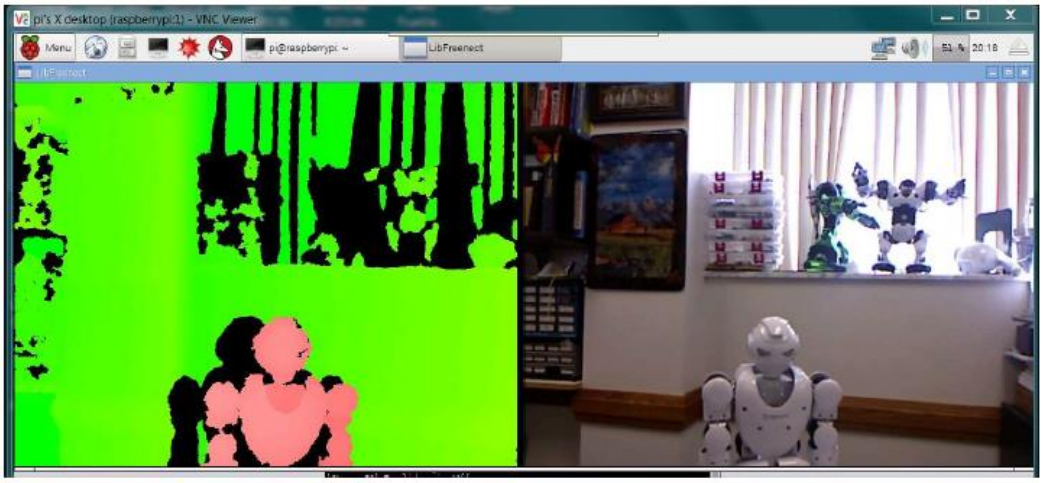
This document is intended to provide guide for Kinect 360 setup with Raspberry Pi.



Accessing the Kinect 360 using the Raspberry Pi

Once the Kinect 360 is connected and powered, one can access the images (RGB+depth) after following installation.

```
sudo apt-get install freenect

sudo apt-get install python-freenect

sudo apt-get install libgl1-mesa-swx11

freenect-glview
```



Virtual Environment

Objective of this document is to collect depth image in real time using python script. Therefore, it is important to setup virtual environment with relevant tools to develop projects with depth information.

```
sudo nano ~/.profile
```

Add the following line at the bottom:

```
VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3.7
```

```
source ~/.profile
sudo pip3 install virtualenv
sudo pip3 install virtualenvwrapper
sudo nano ~/.profile
```

Add the following line at the bottom:

```
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
source ~/.profile
```

Virtual Environment Name: CV

```
mkvirtualenv CV -p /usr/bin/python3.7
source ~/.profile

workon CV
deactivate
```

Installing OpenCV

```
# Clean build directories
rm -rf opencv/build
rm -rf opencv_contrib/build
# Create directory for installation
 mkdir installation
 mkdir installation/OpenCV-"$cvVersion"
 # Save current working directory
 sudo apt -y update
 sudo apt -y upgrade
```

```
sudo apt-get -y remove x264 libx264-dev
## Install dependencies
sudo apt-get -y install build-essential checkinstall cmake pkg-config yasm
sudo apt-get -y install git gfortran
sudo apt-get -y install libjpeg8-dev libjasper-dev libpng12-dev
sudo apt-get -y install libtiff5-dev
sudo apt-get y install libtiff-dev
sudo apt-get -y install libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev
sudo apt-get -y install libxine2-dev libv4l-dev
cd /usr/include/linux
sudo ln -s -f ../libv4l1-videodev.h videodev.h
cd $cwd
sudo apt-get -y install libgstreamer0.10-dev libgstreamer-plugins-base0.10-
dev
sudo apt-get -y install libgtk2.0-dev libtbb-dev qt5-default
sudo apt-get -y install libatlas-base-dev
sudo apt-get -y install libmp3lame-dev libtheora-dev
sudo apt-get -y install libvorbis-dev libxvidcore-dev libx264-dev
sudo apt-get -y install libopencore-amrnb-dev libopencore-amrwb-dev
sudo apt-get -y install libavresample-dev
sudo apt-get -y install x264 v4l-utils

# Optional dependencies
sudo apt-get -y install libprotobuf-dev protobuf-compiler
sudo apt-get -y install libgoogle-glog-dev libgflags-dev
sudo apt-get -y install libgphoto2-dev libeigen3-dev libhdf5-dev doxygen

sudo sed -i 's/CONF_SWAPSIZE=100/CONF_SWAPSIZE=1024/g' /etc/dphys-swapfile
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start


git clone https://github.com/opencv/opencv.git
cd opencv
git checkout $cvVersion
cd ..
git clone https://github.com/opencv/opencv_contrib.git
```

```
cd opencv_contrib
git checkout $cvVersion
cd ..

cd opencv
mkdir build
cd build

cmake -D CMAKE_BUILD_TYPE=RELEASE \
            -D CMAKE_INSTALL_PREFIX=$cwd/installation/OpenCV-"$cvVersion" \
            -D INSTALL_C_EXAMPLES=ON \
            -D INSTALL_PYTHON_EXAMPLES=ON \
            -D WITH_TBB=ON \
            -D WITH_V4L=ON \
            -D OPENCV_PYTHON3_INSTALL_PATH=$cwd/OpenCV-$cvVersion-
py3/lib/python3.5/site-packages \
        -D WITH_QT=ON \
        -D WITH_OPENGL=ON \
                    -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/mo
        -D BUILD_EXAMPLES=ON ..


make -j$(nproc)
make install

 sudo sed -
 i 's/CONF_SWAPSIZE=1024/CONF_SWAPSIZE=100/g
 ' /etc/dphys-swapfile
 sudo /etc/init.d/dphys-swapfile stop
 sudo /etc/init.d/dphys-swapfile start
```

Python Code:

```
import freenect
import cv2
import numpy as np

while True:
    depth, timestamp = freenect.sync_get_depth()
    #array,_=freenect.sync_get_video()
    #array=cv2.cvtColor(array, cv2.COLOR_RGB2BGR)
    depth = depth.astype(np.uint8)
    blur = cv2.GaussianBlur(depth, (5, 5), 0)
    cv2.imshow('image', blur)
    if cv2.waitKey(10) == 27:
        break

cv2.destroyAllWindows()
```

To install pytorch follow: https://medium.com/secure-and-private-ai-writing-challenge/astep-by-step-guide-to-installing-pytorch-in-raspberry-pi-a1491bb80531

More help on freenect: https://naman5.wordpress.com/2014/06/24/experimenting-withkinect-using-opencv-python-and-open-kinect-libfreenect/

7.3 RGB+ Depth Classification Routine

```
# Import packages
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util
import freenect
import frame_convert2

#############################################################

cv2.namedWindow('Depth')
cv2.namedWindow('RGB')
keep_running = True
rgb = []
depth = []

def display_depth(dev, data, timestamp):
    global keep_running
    global depth
    depth=frame_convert2.pretty_depth_cv(data)
    print(depth.shape)
    cv2.imshow('Depth', depth)
    if cv2.waitKey(10) == 27:
        keep_running = False


def display_rgb(dev, data, timestamp):
    global keep_running
    global rgb
    rgb=frame_convert2.video_cv(data)
    print(rgb.shape)
    cv2.imshow('RGB', rgb)
    if cv2.waitKey(10) == 27:
        keep_running = False


def body(*args):

    if not keep_running:
        raise freenect.Kill

print('Press ESC in window to stop')

freenect.runloop(depth=display_depth,
                 video=display_rgb,
                 body=body)

cv2.destroyAllWindows()

lut = np.zeros((256, 1, 3), dtype=np.uint8)
```

```
lut[:, 0, 2] = list(range(0,256))

############################################################
"""
for i in range(5,-1,-1):
    print("Count down: {}".format(i))
    time.sleep(1)
"""
time.sleep(1)
#function to get RGB image from kinect
def get_video():
    array = freenect.sync_get_video()[0]
    array = cv2.cvtColor(array,cv2.COLOR_RGB2BGR)
    return array

#function to get depth image from kinect
def get_depth():
    array=frame_convert2.pretty_depth_cv(freenect.sync_get_depth()[0])
    #array = freenect.sync_get_depth()[0]
    array = array.astype(np.uint8)
    array=cv2.cvtColor(array, cv2.COLOR_GRAY2BGR)
    array=cv2.LUT(array, lut)
    #array=cv2.applyColorMap(array, cv2.COLORMAP_HOT)
    #array=cv2.cvtColor(array, cv2.COLOR_GRAY2BGR)
    return array

# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located
in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different
than detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different
than labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for
displaying detected objects',
                    default=0.5)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH.
If the webcam does not support the resolution entered, errors may occur.',
                    default='640x480')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to
speed up detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu
```

```
# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime,
else import from regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name,
otherwise use default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object
detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load the label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model" from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
    del(labels[0])

# Load the Tensorflow Lite model.
# If using Edge TPU, use special load_delegate argument
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,

experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
```

```
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

time.sleep(1)

#for frame1 in camera.capture_continuous(rawCapture,
format="bgr",use_video_port=True):
while True:

    # Start timer (for calculating frame rate)
    t1 = cv2.getTickCount()

    # Grab frame from video stream
    #frameD=frame_convert2.pretty_depth_cv(freenect.sync_get_depth()[0])
    #frame=frame_convert2.video_cv(freenect.sync_get_video()[0])
    frame = get_video()
    time.sleep(0.02)
    frameD = get_depth()
    time.sleep(0.02)
    #frameD=frame ##########################################################
    #frame=frameD
    # Acquire frame and resize to expected shape [1xHxWx3]
    frame_rgb = frame.copy()
    frame_resized = cv2.resize(frame_rgb, (width, height))
    input_data = np.expand_dims(frame_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as
input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding
box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[1]['index'])[0] # Class
index of detected objects
    scores = interpreter.get_tensor(output_details[2]['index'])[0] #
Confidence of detected objects
    #num = interpreter.get_tensor(output_details[3]['index'])[0]  # Total
number of detected objects (inaccurate and not needed)

    # Loop over all detections and draw detection box if confidence is above
minimum threshold
    for i in range(len(scores)):
```

```
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(frameD, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
################################

        #Add depth information display
            mask = np.zeros((frameD.shape), dtype=np.uint8)

            # define points
            pts = np.array(
[[[xmin,ymin],[xmax,ymin],[xmax,ymax],[xmin,ymax]]], dtype=np.int32 )
            cv2.fillPoly(mask, pts, (255,255,255) )

            # get color values
            values = frameD[np.where((mask == (255,255,255)).all(axis=2))]
            values = np.array(values)
            values = values.reshape(-1)
            print('RGB mean: %d' % int(np.mean(values)))


            # Draw label ########################
            object_name = labels[int(classes[i])] # Look up object name from
"labels" array using class index
            label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example:
'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw
label too close to top of window
            cv2.rectangle(frameD, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) #
Draw white box to put label text in
            cv2.putText(frameD, label, (xmin, label_ymin-7),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

    # Draw framerate in corner of frame ########################
    cv2.putText(frameD,'FPS:
{0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,
0),2,cv2.LINE_AA)

    # All the results have been drawn on the frame, so it's time to display
it.
    cv2.imshow('Object detector', frameD) ########################

    # Calculate framerate
    t2 = cv2.getTickCount()
    time1 = (t2-t1)/freq
    frame_rate_calc= 1/time1
```

```
    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

# Clean up
cv2.destroyAllWindows()
```

7.4 Edge Detection Methods

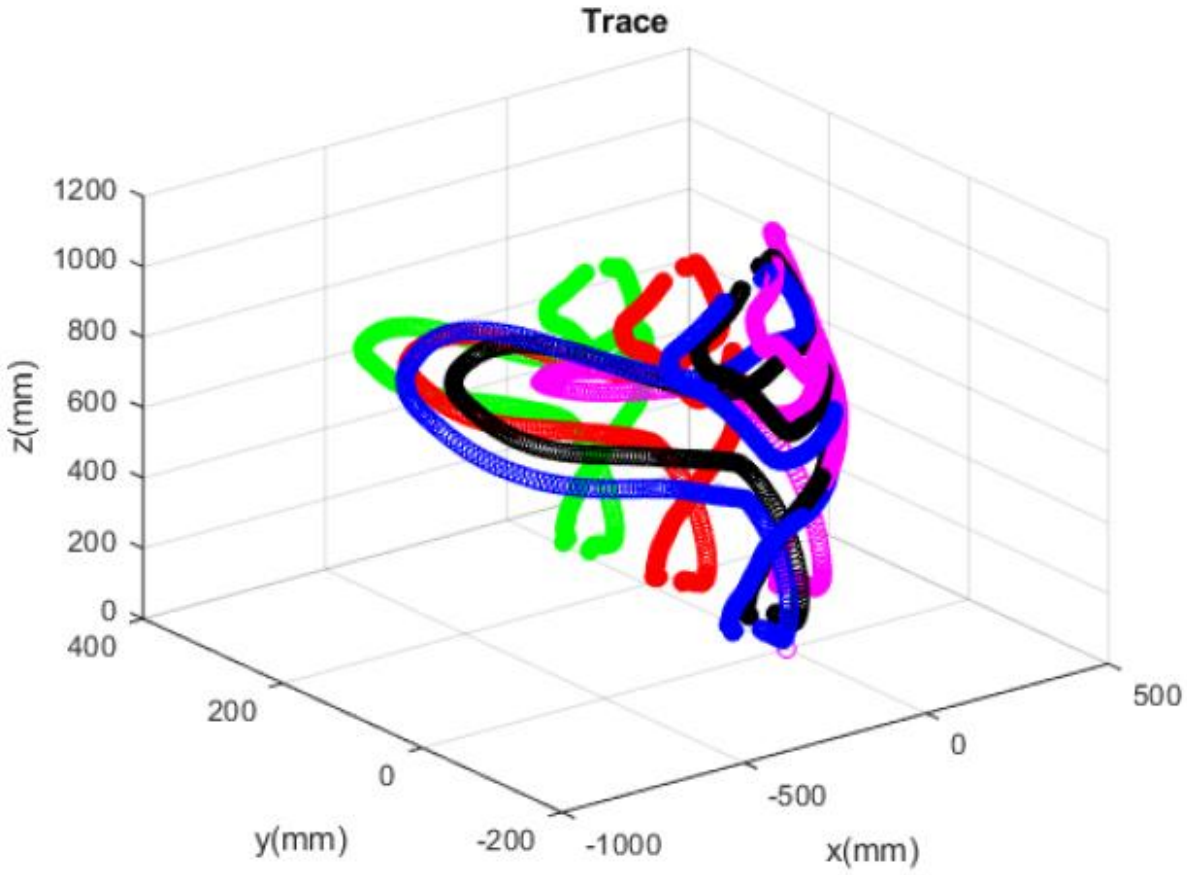
Sobel

BW = edge(I,'Sobel') detect edges using the Sobel method. This method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I is maximum.

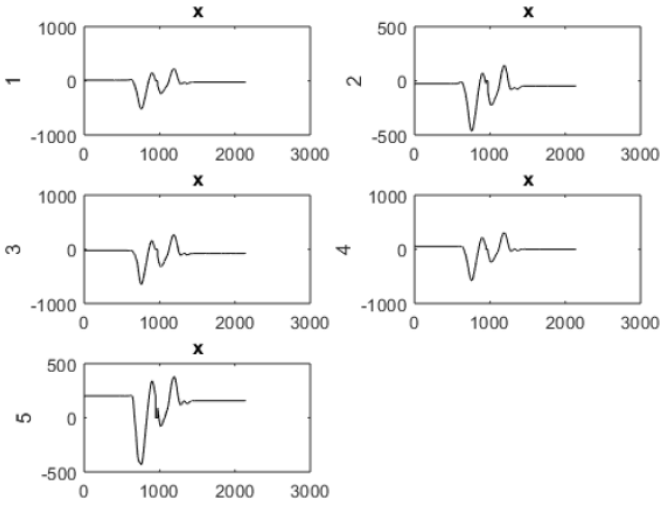
7.5 Vicon - .m files

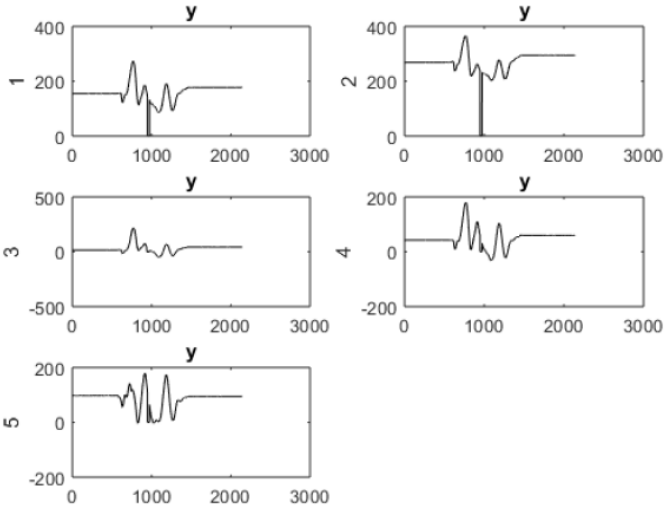
Vicon-motion capture data process

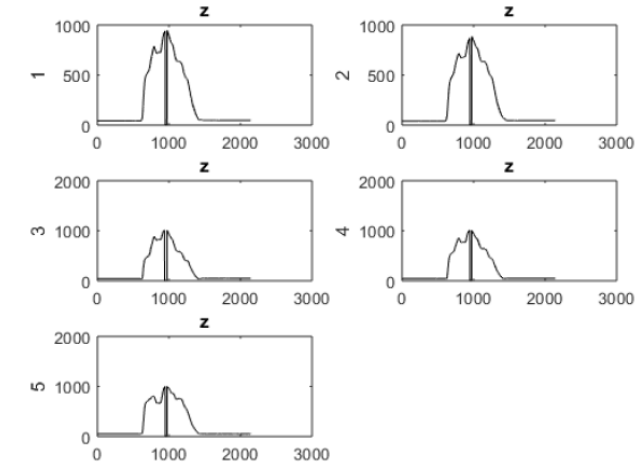
```
% Reset
clc; clear all; close all;
marker=5;
% Trimmed data read
data=csvread('data.csv');
% Define canvas
% axis([-1000,500,-200,400,0,1200]);
% xlabel('x(mm)');
% ylabel('y(mm)');
% zlabel('z(mm)');
% title('Motion');
% grid on;
% hold on;
% Animation
%for i=1:length(data)
%
 plot3(data(i,1),data(i,2),data(i,3),'ro',data(i,4),data(i,5),data(i,6),'go'
%    hold on;
%    pause(0.025);
%end

% Final
figure()
plot3(data(:,1),data(:,2),data(:,3),'ro',data(:,4),data(:,5),data(:,6),'go',
title('Trace');
xlabel('x(mm)');
ylabel('y(mm)');
zlabel('z(mm)');
hold on;
grid on;
```



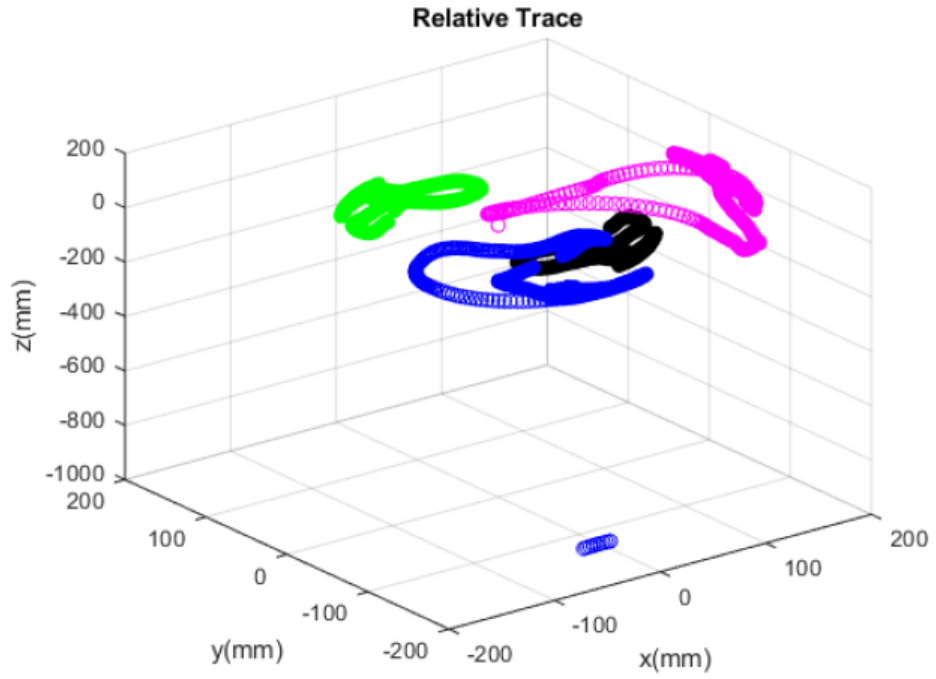

```
figure()
for i=1:marker
    subplot(3,2,i)
    plot(data(:,3*i-2),'k-');
    ylabel(i);
    title('x');
end

figure()
for i=1:marker
    subplot(3,2,i)
    plot(data(:,3*i-1),'k-');
    ylabel(i);
    title('y');
end


figure()
for i=1:marker
    subplot(3,2,i)
    plot(data(:,3*i),'k-');
    ylabel(i);
    title('z');
end
```


x
1
2
3
4
5


y
1
2
3
4
5


z
1
2
3
4
5

```
figure()
plot3(data(:,4)-data(:,1),data(:,5)-data(:,2),data(:,6)-
data(:,3),'go',data(:,7)-data(:,1),data(:,8)-data(:,2),data(:,9)-
data(:,3),'bo',data(:,10)-data(:,1),data(:,11)-data(:,2),data(:,12)-


data(:,3),'ko',data(:,13)-data(:,1),data(:,14)-data(:,2),data(:,15)-
data(:,3),'mo');
title('Relative Trace');
xlabel('x(mm)');
ylabel('y(mm)');
zlabel('z(mm)');
hold on;
grid on;
```




```
figure()
for i=1:marker-1
    subplot(2,2,i)
    plot(euclid(data(:,1:3),data(:,3*i+1:3*i+3)),'k-');
    ylabel(i+1);
    title('distance');
end
```