Chapter 6

Approach for Mobile Robot Navigation and Tracking using Vision Sensor in Dynamic Environment

6.1 Introduction

In recent years, various approaches have been developed to avoid obstacles for mobile robot navigation in dynamic environments. The aim of this work is to develop a methodology for tracking a mobile robot when the environment has dynamic obstacles and extract new information about the mobile robot and path followed, during real-time exploration. Most of the mobile robots are required to navigate in unstructured unknown environment without human assistance and should use several sensors (LIDAR, Camera etc.) to accomplish the task. For above environment conditions robots must navigate in natural and human environments in order to achieve applications like package delivery, cleaning, surveillance, search & rescue. For the mobile robot to navigate in these environments, a map exploration capability is required. The reason for map building is to localize the mobile robot and the goal. In addition to this task, the robot should be able to plan a path from the current position to the target position when the obstacles are moving in the environment with certain velocity. In this work, well-known D* lite algorithm has been implemented for path planning and obstacle avoidance in dynamic environment. In addition, a tracking-learning-detection algorithm (TLD) is implemented to track the mobile robot while performing above said task. The TLD framework [Haifeng & Xiaoqun (2013)] integrates the tracker with the detector via the learning process. It solves a long-term online tracking issue with minimal prior information. The said approach focuses on speed, precision efficient process [Kalal et al. (2010), Park et al. (2015)] and best suited for real-time applications with limited computational power. Whereas KLT tracking method used earlier for tracking purpose in chapter 5, uses points between frames resulting in a sparse field of motion. In this method, feature points are sampled within the object bounding box from a rectangular grid. Based on

motion field the bounding box displacement and scale change are estimated. The KLT method needs comprehensive training data sets and computationally expensive to train. Hence, this approach cannot be applied for tracking of mobile robot in an online setting. To avoid above problems, a fast and accurate detector should be used that localize the patches of the online model and change its decision boundary effectively. Thus, TLD tracking method tracks the object in a video that may include frame cuts, sudden appearance changes and long-lasting occlusions with respect to KLT tracking method.

The intent behind the tracking is to monitor the path followed by the robot and determine whether the real time path suggested by the planner is being followed. For this task, the mobile robot is being observed by an overhead camera. Current work presents an approach where a mobile robot is navigating in an unstructured dynamic environment being tracked using overhead vision sensor and the tracked paths are de-noised using Kalman Filter.

The important sections of this chapter are presented in following ways. The system overview concerning (Kobuki) differential drive mobile robot for real time implementation is discussed in Section 6.2 along with software setup details. In Section 6.3, flow chart to explain the steps followed for mobile robot navigation is presented with experimental procedure in real-time implementation. Subsequently in section 6.4 different obstacle scenarios are created and. implementations of TLD based tracking of mobile robot navigating in a dynamic environment is discussed. The, experimental results and analysis are also presented. Finally, the work is concluded in section 6.5 with the major findings and necessary direction for future work.

6.2 System Description

The mobile robot used in chapters 3-5, was developed in house and had some limitations to implement the path planning in dynamic environment. One of the important limitations was the in-built ultrasonic sensor array and the stereo vision camera for obstacle avoidance. In earlier experiments, mobile robot used was having problems such as inaccurate wheel diameter due to manufacturing tolerances, inaccuracies in actuator drive systems and delays in sensor interfacing. These inaccuracies were dealt through systematic calibration process and were very time consuming. Thus, to alleviate these difficulties the experiments were planned using a commercially available differential drive mobile robot Kobuki which has been shown in Figure 6.1.

Figure 6.1: Mobile robot (KOBUKI) with vision sensor

The said mobile robot has two differential wheels and two castor wheels. It includes sensors for proximity, wheel encoders. In addition, its base can provide power to externally operated sensors such as Kinect, ultrasonic sensors, infrared sensors, laser scanners, other cameras, etc. [Kobuki Documentation Release 2.0, (2017)]. The robot is interfaced with ROS working on an Intel(R) Core(TM) i5-3470 CPU@3.20GHz, 16GB RAM desktop where the task processing became easier. Along with Compaq-HP s009TU fitted as on-board Processing Unit on the Kobuki with an Intel Core i3-4100U Processor and 4 GB DDR3 RAM. The robot ran on Ubuntu 16.04.5 LTS platform with ROS Kinetic configured. The data connection with the Kobuki is interfaced over a USB 2.0/ RX/TX pins on the base. The useful functional and hardware specifications for Kobuki are given below.

Functional Specification for Operation:

- Maximum translational velocity: 70 cm/s
- Maximum rotational velocity: 110 deg/s
- Provision of precise odometry by a 3-axle gyro and high-resolution wheel encoder.
- The imbedded board can be mounted via the Receiver/Transmitter serial port and USB.
- 3 hours of simple battery operation and more than 7 hours of large-size battery operation.
- Payload: 5 kg (hard floor), 4 kg (carpeted floor)
- Docking: within a $2m \times 5m$ area in front of the docking station

Hardware Specification:

- PC Connection: USB or via RX/TX pins on the parallel port
- Odometry: 52 ticks/enc rev, 2578.33 ticks/wheel rev, 11.7 ticks/mm
- Bumpers: left, center, right
- Expansion pins: 3.3V/1A, 5V/1A, 4×analog in, 4×digital in, 4×digital out
- Battery: Lithium-Ion, 14.8V, 4400 mAh.
- Diameter: 351.5mm / Height: 124.8mm / Weight: 2.35kg
- Compaq-HP s009TU fitted as on-board Processing Unit

The vision system is used predominantly for sense of direction, obstacle avoidance and object recognition. For the Kobuki, a stereo ZED camera was used which is very much similar to eyes of the mobile robot. The ZED camera require graphic driver to function correctly, using a NVIDIA 940-MX card that produced required visual results. Further, advantage of stereo ZED camera is that the performance of handling the point clouds and the depth images can be varied as per the requirements. Most of the calibration phase in robotic applications is to precisely calculate the location coordinates and the activity of the camera image pixel, which quantifies the object to be identified based on a number of image examples. The camera parameter like focal length, optical center, coefficients of radial distortion and orientation, estimation method is called calibration of the camera. The goal of the calibration process is to find the 3×3 matrix k, the 3×3 rotation matrix R, and the 3×1 translation vector t using a set of known 3D points (X_w, Y_w, Z_w) and their corresponding image coordinates (u, v) [Zhang (2016)].

Figure 6.2: Real-time test environment

The dynamic environment in which the mobile robot moves is defined in a 2D plane and shown in Figure 6.2. A vision sensor (iball roboK20) is mounted overhead at a distance of 365cm from ground to track the robot. The images are acquired with the help of this sensor with a resolution of 640×480 pixels. The overhead camera is connected to a laptop which is kept away from the mobile robot to provide data to the ROS computational framework. The laptop used for experiments has 1.7 GHz Intel Core i3-4005U processor and Intel HD Graphics 4400 and has Ubuntu 16.04 operating system.

6.2.1 Determination of the Pose of KOBUKI

The mobile robot location with respect to its initial pose should be answered, given that the initial scan is performed in its immediate environment. The pose estimation require two translation components and a rotational component, so that the robot retains an internal estimation of its location and orientation in the world and for a robot confined to the ground: $[x \quad y \quad \theta]^T$. There are few techniques such as wheel odometry, laser scan, and visual odometry which are dependent on the type of sensors used to record the mobile robot position estimation. Odometry is essentially a method that identifies the robot position which can be classified (i) odometry using internal sensors which derives the position from speed, captured by wheel encoders and the steering angle; (ii) odometry using external sensors such as cameras, infrared sensors and ultrasonic sensors are commonly used. By tracking the distinctive features of different images in a sequence, it can be estimate the inter frame transformations (rotation and translation). This is also known as visual odometry and is used on various mobile robots to estimate pose in a global frame. In the case of internal odometry: internal sensors like accelerometers, gyroscopes are used for measurements of the robot to determine pose, rotation or translation. A final position is obtained with respect to the number of rotations of the wheel relative to the pre-defined position, orientation and height. Wheel odometry falls under this category which uses wheel encoders and is supposed to drive the robot along the arc. The number of revolutions is estimated based on the number of ticks which are given by wheel encoders per revolution. The ticks indicate just a fraction of the rotation will increase or decrease, depending on whether the robot is on/off and the distances are determined by the radius of the wheels. The data obtained from these sensors is used for creating an internal map for the robot. This internal map is updated as the robot explores its surroundings. Thus the odometry information is used for determination of the optimal paths which will be used during implementation of path planning algorithm i.e. D* lite algorithms.

6.2.2 Software's for KOBUKI

The following steps and softwares were used in the computer to run the mobile robot:

1. ROS (Robot Operating System - Kinetic version.2019): The ROS framework is most commonly used open source active robot platform that provides abstraction for running and communicating with independent components of the robot. The steps used for interfacing the Kobuki with ROS framework is given below

   Step 1 Install Kobuki: > sudo apt-get install ros-kinetic-kobuki ros-kinetic-kobuki-core

   Step 2 Set Udev Rule: $ rosrun kobuki_ftdi create_udev_rules

   Step 3 Launch Kobuki: > roslaunch kobuki_node minimal.launch-screen

   Step 4 Examine the topics: $ rostopic list

   - Real Time Appearance Based Mapping (RTAB MAP) using ROS: This feature is used for creating maps with the help of the vision sensors, in this case ZED-camera. RTAB map takes the point cloud generated by the vision sensors and appends all the point clouds in order to produce a 3D map of the environment captured by the vision sensor. There are two different types of maps that can be used for mapping procedure i.e. (i) depth projection map approach and (ii) virtual scan based map update approach.
   - RViz (ROS Simulator): It is a simulator in which helps in visualizing the sensor data in the 3D environment. To represent and visualize information in the real domain problem, it is used as a visualization tool which includes geometry synthesis, distance field representation and implicit surface reconstruction. In this work, the sensor data are used to visualize the static and dynamic obstacles in the workspace.

2. MATLAB (version-2019b) for processing the images captured by iball camera for tracking purpose

3. Python (ROSPY) version 3.6.6 for interfacing with ROS and MATLAB

6.3 Overview of Experimental Implementation

In this work, virtual scan based map setting has been used to get a depth image from ROS package, which converts the depth images to possible object and this was compatible with the ROS Navigation stack and the D* lite map updating process. In addition, a graphical driver, for NVIDIA 940 MX graphic card, was used to operate the ZED camera which resulted in improved performance. The only drawback is that neither robot could be operated remotely from a separate device, because the graphics driver requires power supply and practically running at very high processing loads. The robot movement in the environment was thus limited. But the same test for implementation of D* lite was carried out in a simple Rviz tool. In this case the mobile robot will escape the static and dynamic obstacles and find the shortest path to the point of starting position. The steps given below explain the procedure for experimental implementation of algorithm.

Steps used for experimental implementation:

Step-1: The ROS nodes for mobile robot and vision sensor are launched.

Step-2: The depth image topic is published by the vision sensor into virtual machine.

Step-3: A TF between vision sensor and mobile robot is created.

Step-4: The ROS navigation stack and setting the global planner to D* lite are run.

Step-5: RTAB MAP is launched and mapping mode is initiated.

Step-6: The Goal selection python script is run.

Step-7: The results are published and displayed.

The flowchart for implementation is presented in Figure 6.3.

6.3.1 Flow Chart for Mobile Robot Navigation

Navigation is a process that a mobile robot needs to perform correctly without losing or collision with barriers, to move safely from one location to another. This includes reactive approach to circumstances such as collisions and path planning for object avoidance. The navigation system performs three main functions; object tracking, path planning and obstacle avoidance. At the initial position, the vision sensor on the mobile robot performs a search for the obstacle and destination; if detected moves towards it and tries to avoid obstacles and go towards for the destination; if not detected plans a path to move to a new position in order to continue the search of destination. Mobile robot tracking using over-head camera and path planning strategies will be explained by using flowcharts shown in Figure 6.3.

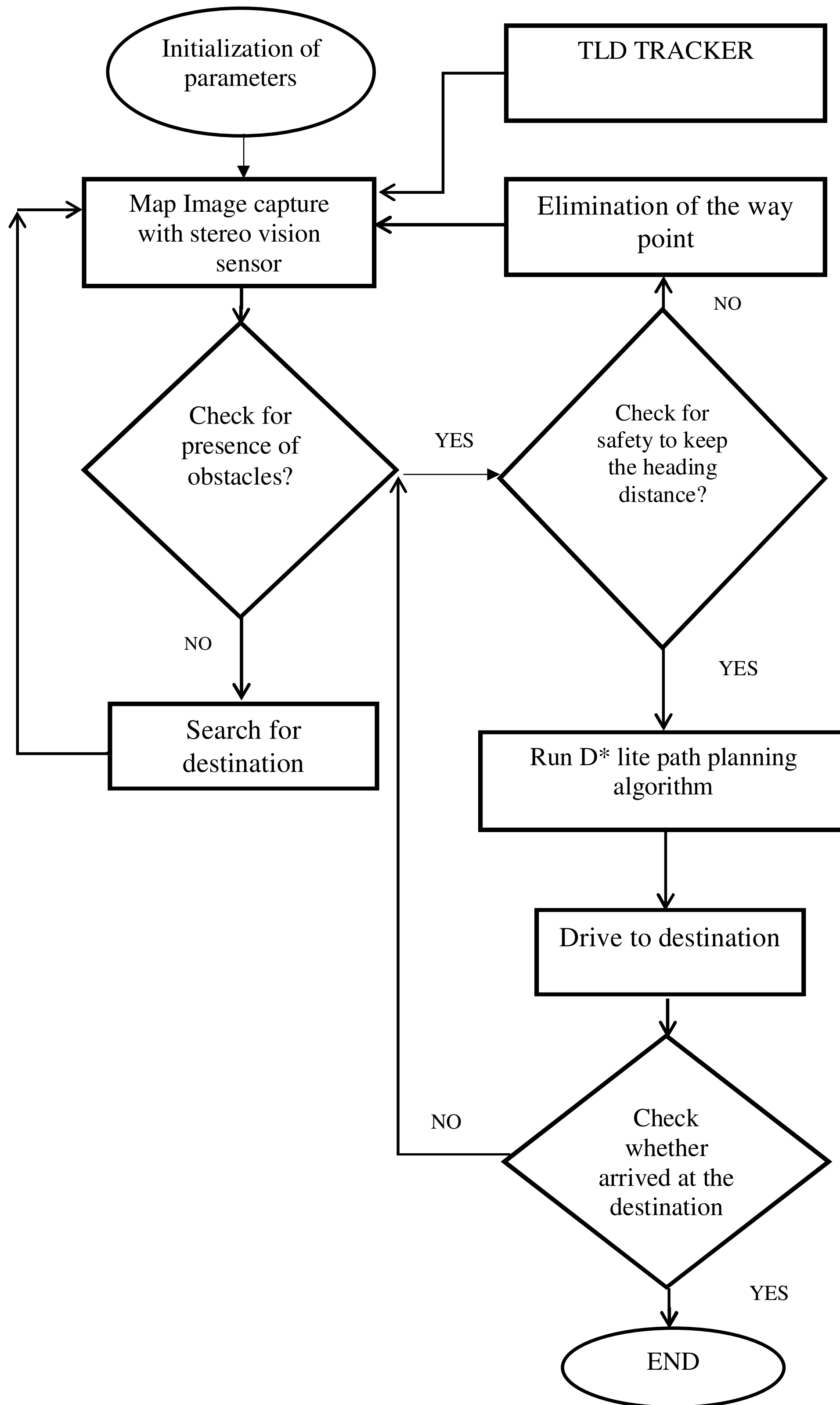


Figure 6.3: Flow chart for mobile robot navigation

6.3.2 D* Lite Path Planning Algorithm

Nearly all environmental information is unknown to the mobile robot before executing any path planning task. To explore in a dynamic environment local path planning helps in avoiding the current local obstacles. Incremental search path based on the unknown field as a free space is the central part of the D* lite algorithm. For local path planning D* lite algorithm [Han-ye et al. (2018)] has been utilized so that the robot avoids current local obstacles which were not present earlier. The target point of local path planning is the point closest to the global path planner which is at the edge of local navigation region. Once the goal is assigned to the mobile robot by the global planner, then the next step is to reach the goal using goal selection algorithm. But it may so happen that the goal point may not be reachable in a straight path, or there may be an unknown dynamic obstacle which comes in the path of the robot, preventing it from reaching the goal. During the testing and implementation of this algorithm on the ROS navigation stack, it is found that the ROS navigation stack is not directly compatible with the dynamic path planning approaches. As such the algorithm has to be recoded as an external path planner for the navigation stack. The approach for path planning experiments, which uses global planner, only provides information to move in straight line. Now as the navigation stack receives the goal point from the Global planner this message is sent to the external planner instead of directly being executed.

The external planner then determines the intermittent goals and sends these to the ROS navigation stack and then the global planner is forced to reach these intermittent goals. Once an intermittent goal is reached then a next intermittent goal is given and so till the robot reaches the final goal. In real implementation, it is assumed that a path observed by mapping $=\{(X_1,Y_1),(X_2,Y_2),(X_3,Y_3),....(X_t,Y_t),\}$ where (X_1,Y_1) is the current position and (X_t,Y_t) is the ultimate goal. Let the mobile robot starts from (X_2,Y_2) for each (X_i,Y_i), and then first check if this point is observed; then checks if $(X_i - X_{i-1}, Y_i - Y_{i-1}) = (X_{i-1} - X_{i-2}, Y_{i-1} - Y_{i-2})$. If both conditions are true, it updates the temporary target to be this point. This helps us to send intermittent goal to the ROS nav-stack.

The rest of this section starts with some context in D* lite algorithm path planning which is explained briefly the development of the D* lite from the classic A* algorithm search. It is needed to understand generic graph search and A* algorithm search before defining D* lite algorithm technique for mobile robot navigation. Thus, it will incorporate some of the terminology found in literature that helps explain how the D* lite algorithm is created [Koenig & Likhachev (2002)]. The obstacles information data are stored in the cost maps used for short range planning and obstacle avoidance. With the cost estimate update, the robot moves to the end of the target and calculates continuously the shortest paths between the current vertex and a target vertex. The function 'g' explains the actual costs of traverse the optimal path to the current node from the initial state and the function '*h*' is an estimation of the additional cost from the current node to the target. The combined functions '*g*' and 'h' represent an estimation of the cost of carrying the current node from the initial state to the target along the path. Let '*S*' represent the finite set of the graph vertices. Let Succ(*s*) □ *S* indicates the set of vertex successors to S and in similar way, Pred(*s*) □ *S* denotes the vertex predecessors to S. In the LPA* algorithm [Koenig et al. (2004)], an analogue of the *g*(*s*) value of A* algorithm search is presented for each vertex. The rhs-values are one-step-looking values based on the g-values and therefore theoretically better informed than the g-values, which are the second type estimation of the start distances. The 'rhs' values always satisfy the following relationship in Equation 6.1.

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{\text{start}} \\ \min_{s' \in \text{Pr}ed(s)} (g(s') + c(s', s)) & \text{otherwise} \end{cases} \quad (6.1)$$

where, *s*: current vertex; $\text{Pred}(s)$: Predecessors of the current vertex. $c(s', s)$: edge cost to traverse from s' to s. A node s is defined as locally consistent when $g(s) = rhs(s)$ and locally inconsistent when $g(s) \neq rhs(s)$. LPA* maintains a priority queue just as A* but it contains only those nodes which are locally inconsistent. If all vertices are locally consistent, then the g-values of all vertices equal their respective start distances. In this case one can trace back a shortest path from s_{start} to any vertex u by always transitioning from the current vertex (s), starting at (u), to any predecessor that minimizes $g(s') + c(s', s)$ (ties can be broken arbitrarily) until s_{start} is reached.

The first search of LPA* is similar to the A* search. But, LPA* doesn't make all nodes locally consistent when there are obstacles on its path i.e. if any edge costs change. Instead, it uses the heuristics to focus the search and updates only the g-values that are relevant for computing the shortest path. Here the priority queue plays a major role in determining the locally inconsistent vertices, which are then rectified on the basis of the key '$k(s)$' of the vertex which is defined as in Equation 6.2:

$$k(n)=\begin{bmatrix} k_1(n) \\ k_2(n) \end{bmatrix}=\begin{bmatrix} \min\left(g(n),rhs(n)\right)+h(n,\text{goal}) \\ \min\left(g(n),rhs(n)\right) \end{bmatrix} \quad (6.2)$$

Here n or s both denotes a vertex or node. LPA* searches from start-vertex to goal-vertex can be checked to calculate the estimates of g-values. However, D* lite calculates the estimates of the g-values from the goal-vertex to start-vertex. Now when edge-costs in the graph change, it causes a reordering in the priority queue of the LPA* algorithm, which is very expensive as the priority queue contains huge number of vertices for large graphs. The pseudo code for D* Algorithm is provided in the Appendix-(E.1), which can be summarized in following five steps.

Step-1: Initial expansion around the goal with calculation of $g(s)$, rhs(s), $h(s)$. The minimum cost estimation for a starting point of node distance is g(s) and rhs(s). The $g(s)$ value will be recalculated by expanding adjacent grid, because the search direction of the D* lite algorithm is opposite. The value of the g(s) of the forward point is defined by rhs(s) and the $h(s)$ heuristic function is the same as the heuristic function in the A* algorithm.

Step-2: Computation of the Euclidean distance from the current vertex to the start vertex for the heuristic estimation.

Step-3: Establish to find the initial shortest path while finding new vertex that needs to be expanded if the obstacle moves.

Step-4: Computation of expansions around the current vertex as the obstacle move.

Step-5: Determine the shortest path by analyzing surrounding nodes to find the next node to be expanded. Node update as the robot moves in the direction of the goal.

For the experimentation approach to be successfully implemented, the robot uses a frontier exploration (me134_explorer) simulator that separates known regions from unknown regions which is a ROS node from Github [ROS node for Turtle Bot robot (2019)]. After several trials, the final solution is achieved by combining the frontier explorer and a global planner approach, which uses straight line path. In this design, explorer will execute D* lite for path planning and it will select a point on the path, which satisfies that: (i) it can be reached by a straight path. (ii) All the way to reach the point has been explored, as the goal and send it to global planner. If this objective has been accomplished, the explorer will update the map and re-run D * lite algorithm and send a temporary goal to the global planer. In this way, instead of planning the whole path all at once, it breaks down the end goal into a series of temporary goals which can be safely achieved.

6.4 Experimental Results and Analysis

All the programs were coded and run on Python 3.6.6 platform and the graph searches were carried out on a PC with Intel Core i5-8300H and 8 GB DDR4 RAM. The variation in the total time taken and total number of operations were stored for D* lite incremental heuristic search algorithm over a number of grid sizes. Vision sensor is used for 360-degree detection of environment. The experiments are tested in unknown environment with ROS model with 640×480 pixels image size and with a frame rate of 30 fps. The on board ZED camera mounted on Kobuki captured the images and converted it to disparity maps for every 30-degree rotation of the mobile robot. The outputs of the camera are the comparison matrix of disparity $dispmat$, the orientation of the robot for which there is lowest disparity represented by the angle of rotation of the robot, the maximum value of the percentage of free space in that particular orientation, the goal points selected represented as the x and y coordinates. A disparity MAP (Max Value of lowest disparity of 0.9212) is obtained that is published in the depth map. This distance is converted to the possible coordinates for the robot pose and are sent to the action client (refer to ROS navigation stack) which then gives a signal to the /move_base node of the ROS navigation stack, which further send signals to the robot wheels to rotate by a certain number of rotations till the goal is reached.

The ROS rqt_graph package allows users to visualize the ROS computational graph, showing all peer-to-peer communication with ROS that are processing data together. The ROS rqt_graph creates a dynamic graph while system is in running condition which displays the various topics and nodes. The ROS rqt_graph depicting the various topics and nodes is shown below Figure 6.4.

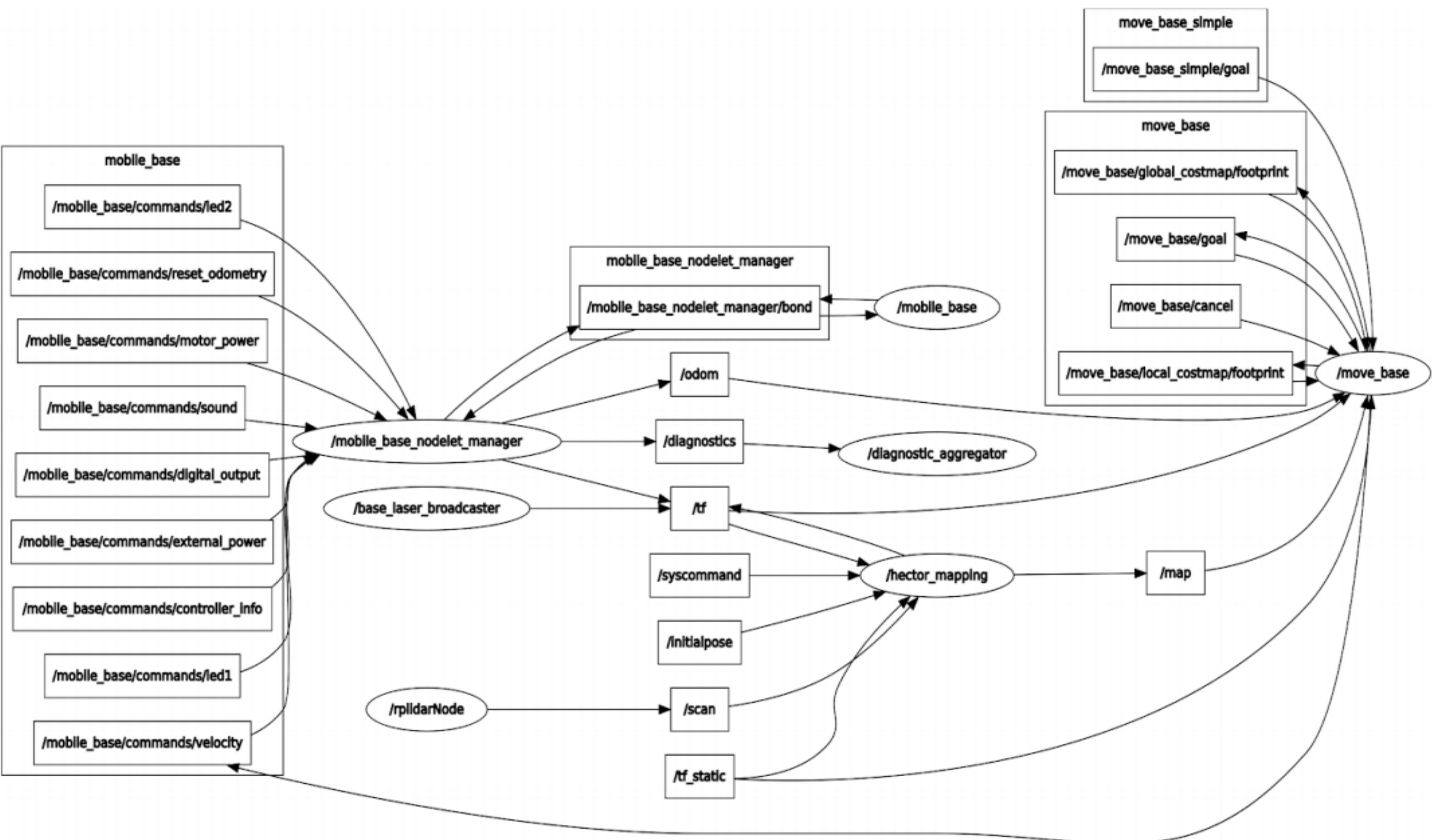


Figure 6.4: Visualization of Rqt_graph with various package topics and nodes diagram

The rqt_graph is a method for ROS debugging identifies the connection between every node and the topic that is communicated. The idea is that a node that wants to exchange information will publish messages on the appropriate topic/topics. The rqt_graph needs to be updated during the exploration of node, since the graph doesn't automatically update. After the update, /move_base node is included in its modified ROS network. Also, the move_base_node publishes a new topic /move_base_simple/goal which is subscribed by the move_base node. The easiest way to visualize publish-subscribe connections among ROS nodes is by using commands are mentioned in Appendix (E.2).

The objective of these tests is to see if the mobile robot can avoid any unexpected dynamic obstacles in its path to the goal. The global cost map is used to plan the entire operational environment and the local cost map is used for short range planning and collision free obstacle. Given a path from current position to the goal, mobile robot selects the temporary goal which has already been observed and can be reached by moving along a straight line path.

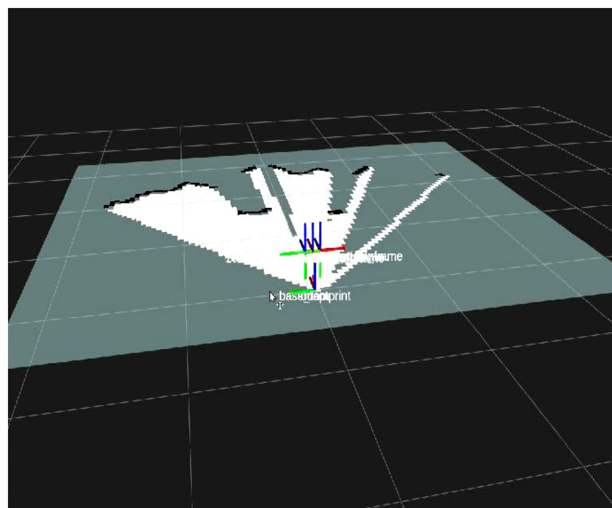


Figure 6.5: Initial scan of the mobile robot environment model

Once the map has been developed, it is now carried out the entire navigation process simulation. Figure 6.5 illustrates a snapshot of the map of the surrounding, which was processed with visualization software. The screen-shot of initial scan of the mobile robot environment real world test run model is shown. The image of the map with white cells indicate that the cells are to be explored and empty; Black cells indicate that cells are explored and occupied; Grey cells indicate that cells are unexplored; and Red dots represent the current sensor scan endpoints.

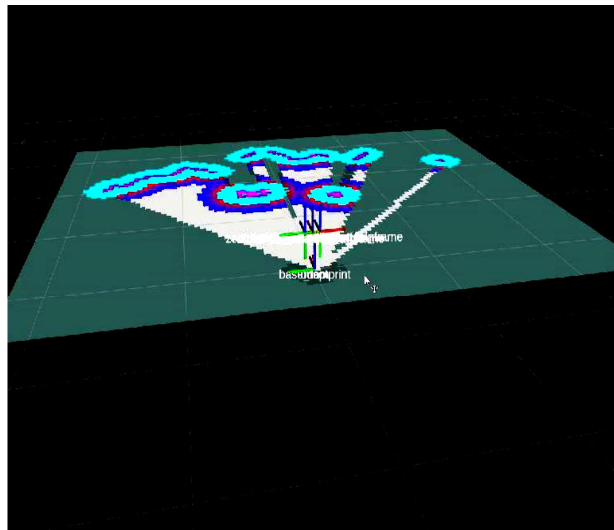


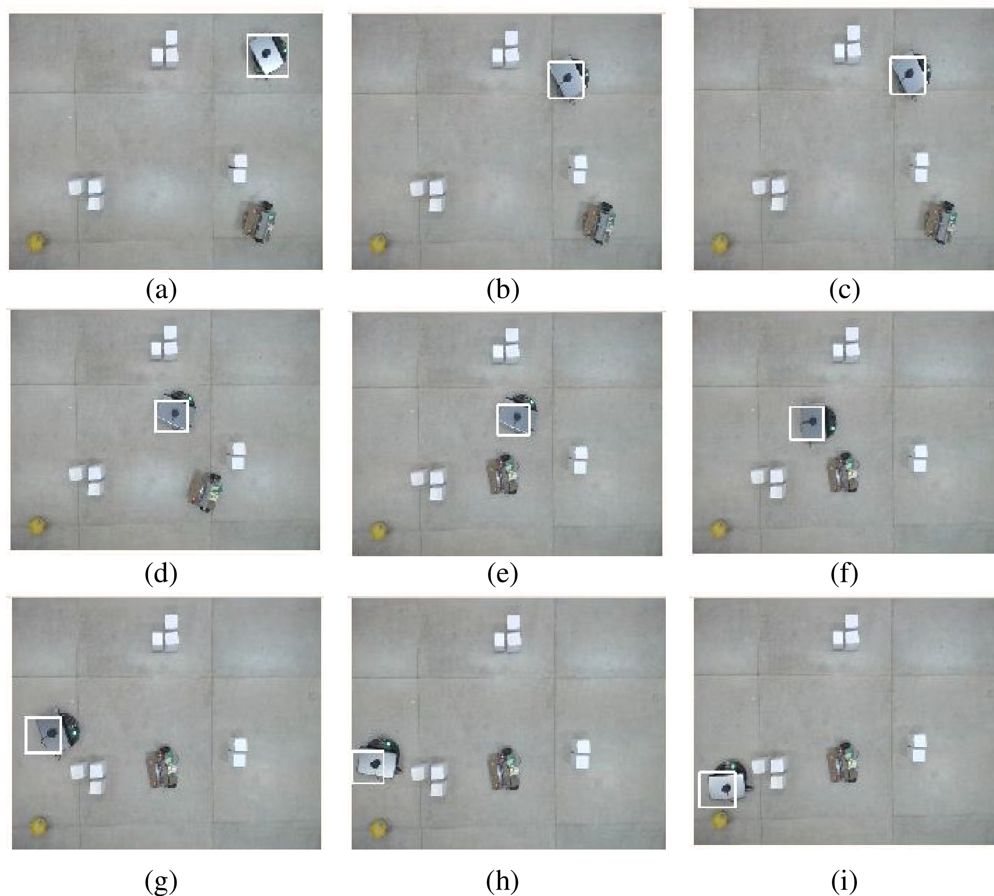
Figure 6.6: Local environment cost-map

Local environment cost-map which store information about obstacles composed of static map layer, obstacle map layer and inflation layer are shown in Figure 6.6. Static map layer interprets the static SLAM map generated for the navigation stack directly. Obstacle map layer includes 2D/3D obstacles are either marked (detected) or cleared (removed) based on robot's sensor data, which has topics for cost-map to be subscribed. During the update portion of each cycle, new sensor data is placed into the layer's cost-map. Inflation layer is where obstacles are filled to calculate cost for each 2D cost-map cell with cost ranging from 0 to 255. In addition, there is a global cost-map and a local cost-map. Global cost-map is generated by filling the obstacles on the map provided to the navigation stack. Local cost-map is generated by filling obstacles detected by the vision sensor in real time.

As such the process of the goal selection begins by robot starting to scan the surrounding area by rotating 360°. While rotating the ZED camera attached to the robot captures rectified RGB images from its camera. These are then sent to the MATLAB code for conversion into disparity maps. The sum of the values of the pixels which are in the different ranges of allowable disparity values is determined when a disparity map is produced. Then the proportions of the pixels which are in the range of lowest disparity are determined as shown in the MATLAB code in the Appendix (E.3 & E.4). The use of Computer vision tool box in MATLAB, combined with the MATLAB ROS interface finally gave the way to create disparity maps.

These tests were performed in both simulation framework and real-time implementation on Kobuki. The results are recorded and the mobile robot avoided static as well as dynamic obstacles. Mobile robot movement data (odometry) in on real-time scenes were obtained. It is observed that the farther the lighting from the robot the greater the distance obtained by the depth image. The distance between the obstacle and the mobile robot is shown in Figure 6.7 (a-i). For the discussed scenario another mobile robot was commanded to move in the environment and treated as dynamic obstacles. In this case a dynamic window approach algorithm is deployed algorithm for local navigation.

Case-I: Test path 1



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 6.7 (a-i): Tracking of mobile robot in test path 1 using TLD algorithm

In test path 1 scene, The TLD tracking is initialized by a bounding box and it estimates the size of the mobile robot in rotation, occlusion, pose shifts, and change of scale correctly. For tracking the mobile robot with diameter 25cm, a bounding box of size 110×90 is chosen which provided effective results at 10 frames per second. In this work,

mobile robot being tracked using TLD algorithm is depicted visually by a surrounding bounding box in the shape of a 'white' rectangle shown in Figure 6.7 (a-i). The size of the rectangle is determined according to the feature points detected near the object. The measured position (location) of the mobile robot is taken to be the centroid of the depicted rectangle in the image. For tracking the mobile robot, a bounding box around the captured image is used for different instance of time. The bounding box has a fixed aspect ratio and is parameterized by its location and scale. Other factors, such as in-plane rotation, are not considered for this experiment.

- The error percentages for tracking were less initially as the mobile robot was having uniform velocity. As the time elapse the tracking error become higher and higher, because the mobile robot encounter static and dynamic obstacles, which resulted in a change in orientation and velocity to negotiate these. Finally, the error becomes highest due to in-plane rotation of mobile robot as the robot was ensuring whether it is surrounded by static/dynamic obstacles.
- The above error can be reduced in by considering in-plane rotations during TLD implementation which can be dealt at later stage and can be kept as a future scope of present work.

Table 6.1: Odometry coordinates w.r.t. time and TLD tracker for test path 1

Co-ordinates/ time (sec)	**2 sec**	**9 sec**	**18 sec**	**27 sec**	**38 sec**	**45 sec**	**58 sec**
Odometry(*x*)	512.19	514.90	489.05	409.71	373.54	130.86	120.30
Odometry(y)	89.30	93.97	113.08	200.95	247.20	268.49	295.14
TLD (*x*)	512	515	479	394.5	358	96	81.5
TLD (*y*)	89	93	105.5	187.5	232.5	234	258

The experimental map area on the ground is approximately 160 cm × 230 cm. The side of each pixel in the image can be taken as 0.302085cm on the ground. Using this assumption, TLD tracking of the frame (x) vs. frame (y) plot was found out from the experiments presented in Figure 6.8.

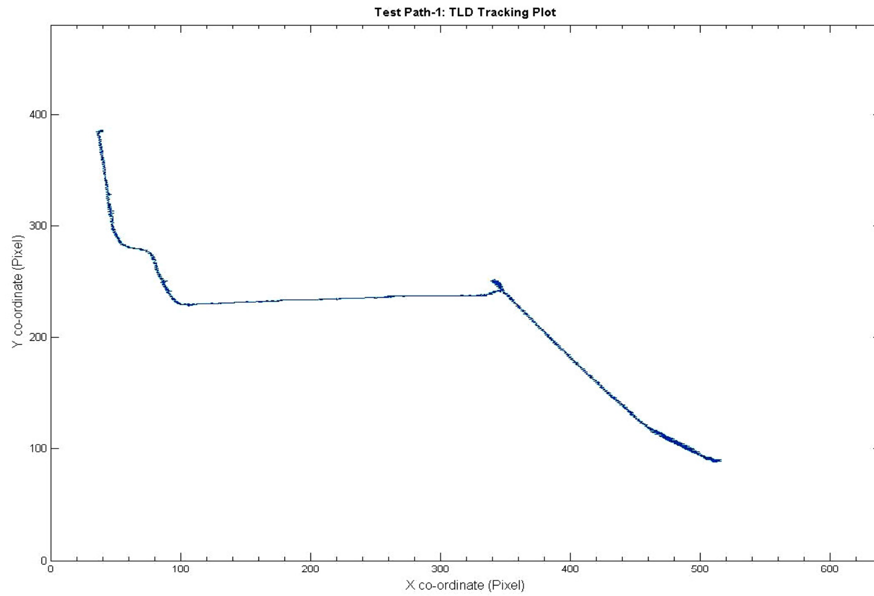


Figure 6.8: Tracking of mobile robot for test path 1 using TLD algorithm

It displays mobile robot tracking path for test path 1 results using TLD algorithm where the measured path (green) has detected features at the top of the object. It estimates the motion of mobile robot under the assumption that the object is visible with limited motion. Several image sequences have been used for experimentation. The results of the computed co-ordinates of TLD tracking with respect to time are shown in Table 6.1 showed better tracking accuracies.

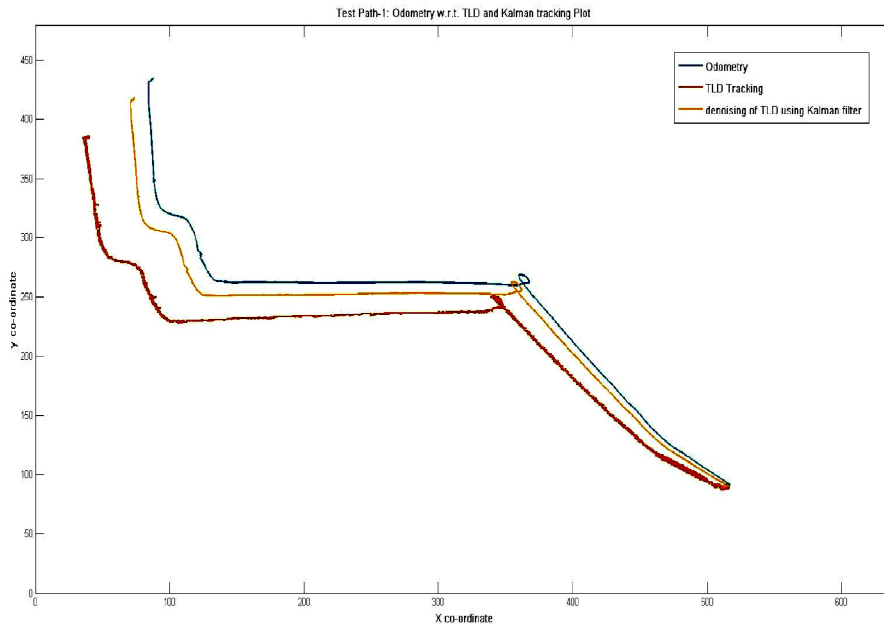


Figure 6.9: Tracking of mobile robot using TLD and denoising of TLD using Kalman filter

The Kalman filter (discussed in Chapters 3 and 5) is a method for combining physical models and sensor data information to estimate the position of the robot, the direction and the speed at which it travels. The modeling in a linear system is typically used for Kalman filter consisting of update prediction stage and update measurement stage. Prediction model includes the actual system and the process noise. The update model includes updating the predicated or the estimated value with the noise of observation. The initial Kalman filter parameter values have been calculated experimentally with $Q = 1.23$, $R = 0.5$, $A = 1.0$ based on empirical data. The motion vectors are used in depicting the 'motion' of the measured position of the mobile robot. The Kalman predicted location (red) in Figure 6.9 has accurately been detected the sudden change of path of mobile robot using the information from previous frames of odometry and TLD tracking. The distinction between the measured and filtered (estimated) path can be observed clearly in Figure 6.9. The path tracked using TLD algorithm and Kalman filter based denoising have been analyzed.

Table 6.2: Error % in TLD coordinates and TLD with Kalman filter coordinates with respect to odometry coordinates for test path 1

Error % after t (sec)							
	t = 2	t = 9	t = 18	t = 27	t = 38	t = 45	t = 58
Odometry w.r.t. TLD (Coordinates)	0.06	0.18	2.28	3.7	3.9	8.9	9.7
Odometry w.r.t. TLD with Kalman filter (Coordinates)	0.06	0.18	1.48	1.7	3.02	4.02	6.42

The error % for the TLD based tracking and Kalman filter based denoising were computed at various positions and for the computation Equation (6.2) and (6.3) are used.

$$\text{Percentage of error} = \frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sqrt{U_{\text{map}} \times V_{\text{map}}}} \times 100 \tag{6.2}$$

where (x_i, y_i) co-ordinates of odometry, (x_j, y_j) co-ordinates of TLD tracking path, $(U_{\text{map}}, V_{\text{map}})$ length and breadth of real time map.

$$\text{Percentage of error} = \frac{\sqrt{(x_i - x_k)^2 + (y_i - y_k)^2}}{\sqrt{U_{\text{map}} \times V_{\text{map}}}} \times 100 \tag{6.3}$$

where, (x_k, y_k) co-ordinates of denoising of TLD using Kalman filter path, (U_{map}, V_{map}) length and breadth of real time map.

The maximum error is of the order 10 % due to unsteady velocity and change in the direction of the mobile robot using TLD algorithm. As per the result shown in Table 6.2, it can be observed that Kalman filter provides less than 6% tracking error in comparison with TLD tracking and Figure 6.10 provide the trends of error % in test path 1. It can be observed that the error percentages were less initially as the mobile was having uniform velocity. As the time elapse the error become higher and higher as the mobile robot encounter static and dynamic obstacles, which resulted in change in orientation and velocity to negotiate those. Finally, the error becomes highest due to in-plane rotation of mobile robot as the robot was ensuring whether it is surrounded by static/dynamic obstacles.

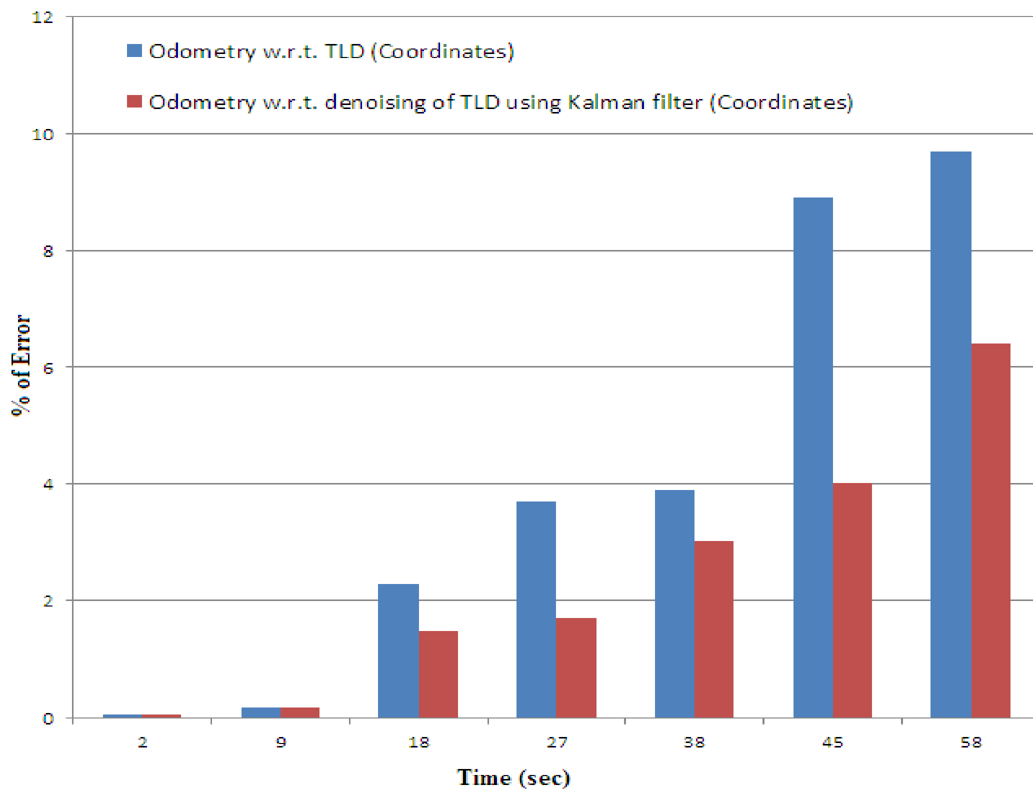
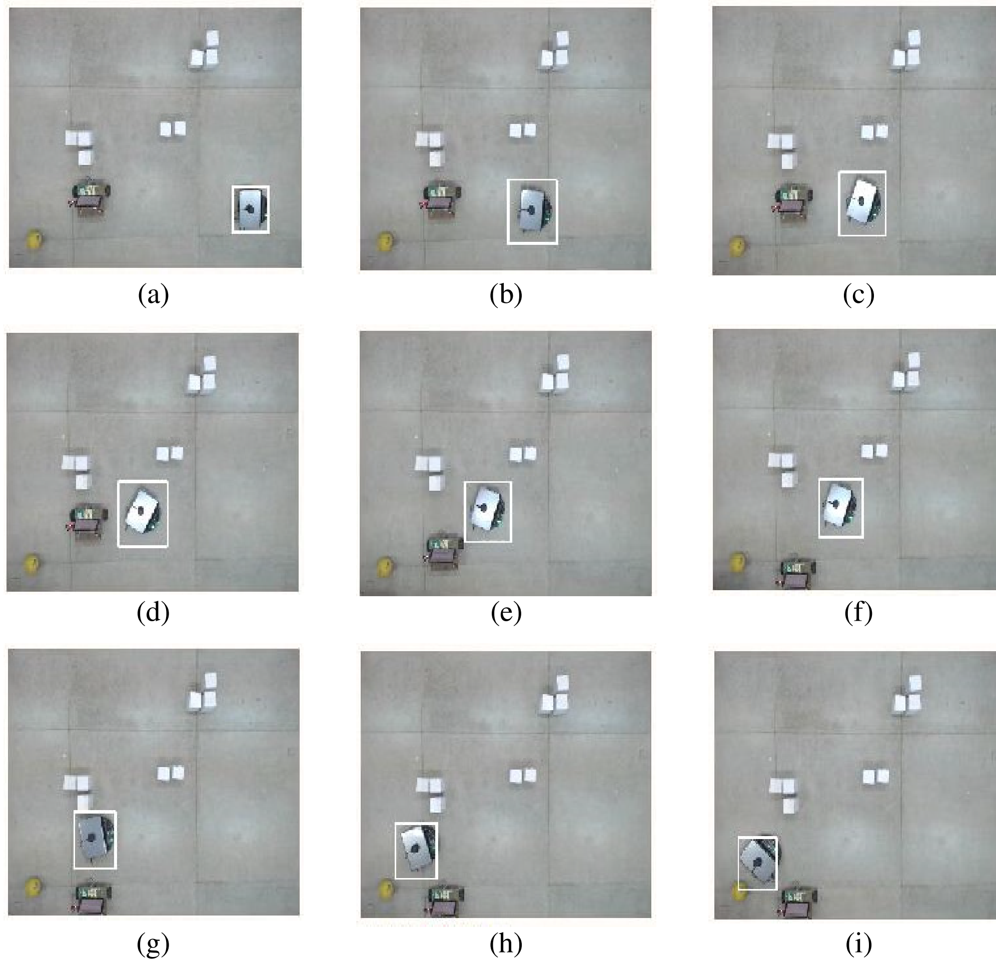


Figure 6.10: Percentage of error for TLD tracking and error after denoising of TLD using Kalman filter for test path 1

Case-II: Test path 2



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 6.11 (a-i): Tracking of mobile robot in test path 2 using TLD algorithm

The results of the computed xy co-ordinates of TLD tracking are presented in Figure 6.12. Therefore, the time interval of two inter-frames is very small in the real-time follow-up phase, and the target travel of consecutive frames can be considered linear. TLD tracker has some shortcoming owing to the lack of illumination and fast turn of mobile robot away from the central feature of the image thus feature points on image may not yield correct tracking. That is the reason why tracked path using TLD algorithm observed to be different from the planned path.

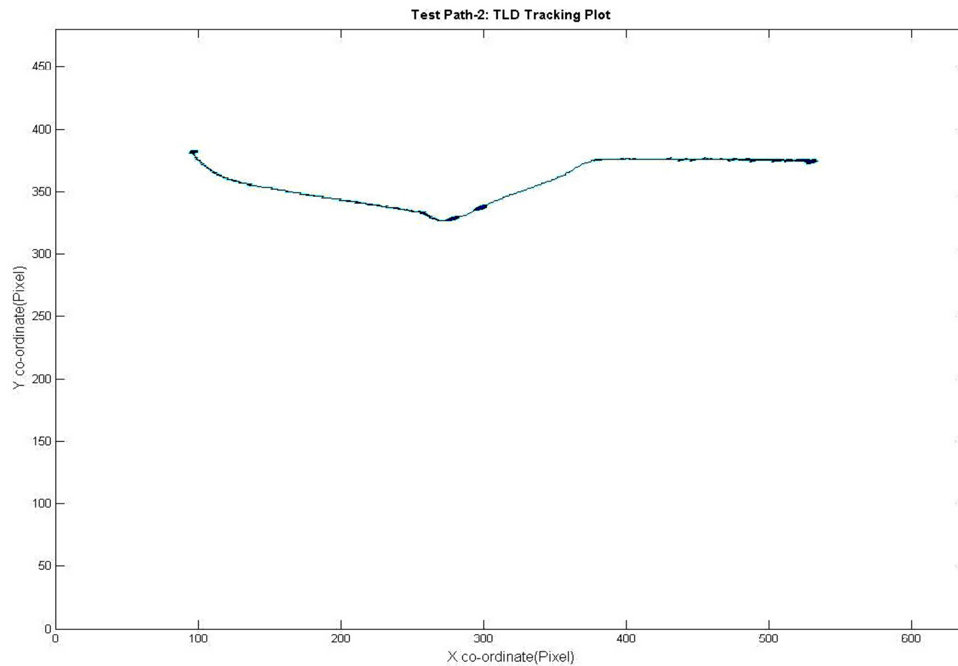


Figure 6.12: Tracking of mobile robot using TLD algorithm for test path 2

Thereafter Kalman filtering algorithm is implemented on the tracked path and consistent covariance bounds are obtained. This implementation leads to the tracking results of mobile robot path closer to the path used for navigation.

Table 6.3: Odometry coordinates w.r.t. time and TLD tracker for test path 2

Co-ordinates /time (sec)	3 sec	9 sec	15 sec	21 sec	27 sec	33 sec	39 sec
Odometry(x)	529.74	503.99	306.34	312.60	317.09	302.74	261.38
Odometry(y)	373.70	378.29	345.26	350.50	354.99	351.38	363.47
TLD (x)	530	501.50	297.5	301	298.5	283	234
TLD (y)	374	375.5	336.50	336.5	338.5	329	337.5

The velocity refers to the displacement of the mobile robot through two consecutive frames. The data from experiments was taken to test and the performance of algorithms. A number of scenes with different time period in Figures 6.7 (a-i) and Figures 6.11 (a-i) were shown to illustrate test path 1 & 2. Comparison of TLD tracker (yellow) with respect to Kalman filtering (Red) is presented in Figure 6.9 and Figure 6.13.

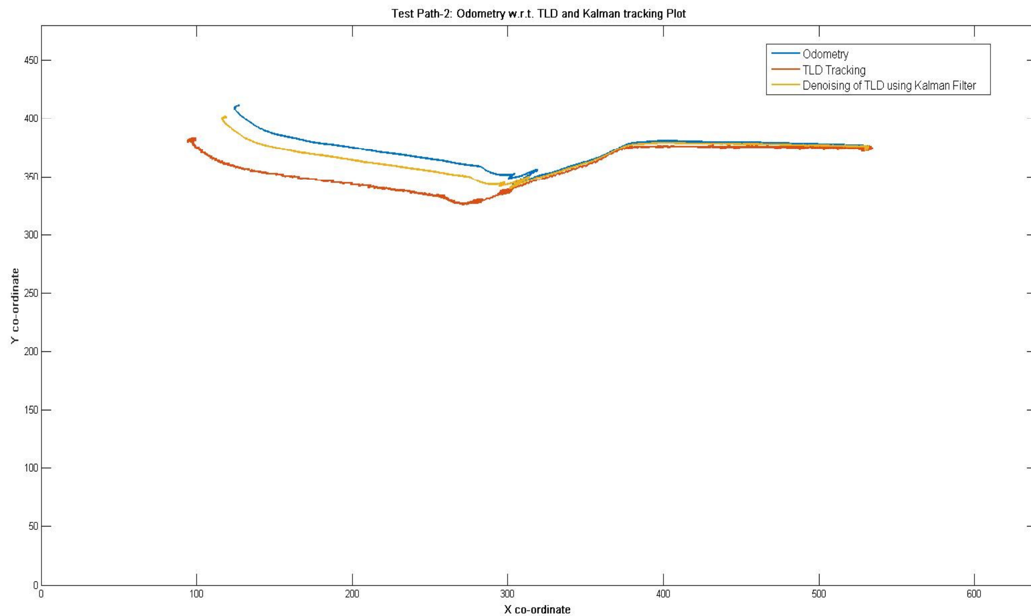


Figure 6.13: Tracking of mobile robot using TLD and denoising of TLD using Kalman filter

Likewise, in the test path 2 shown in Table 6.4 presented the positional error of less than 7% with regard to TLD tracking and 4% after Kalman filter based denoising and the error trend is presented in Figure 6.14 for test path 2. It can be observed that the error percentages were less initially as the mobile was having uniform velocity. As the time elapse the error become higher and higher as the mobile robot encounter static and dynamic obstacles, which resulted in change in orientation and velocity to negotiate those. Finally, the error becomes highest due to in-plane rotation of mobile robot as the robot was ensuring whether it is surrounded by static/dynamic obstacles.

Table 6.4: Error % in TLD Coordinates and TLD with Kalman filter coordinates with respect to odometry coordinates for test path 2

Error % after t (sec)							
	t = 3	t = 9	t = 15	t = 21	t = 27	t = 33	t = 39
Odometry w.r.t. TLD (Coordinates)	0.07	0.67	2.25	3.3	4.5	5.4	6.8
Odometry w.r.t. TLD with Kalman (Coordinates)	0.07	0.67	1.40	2.37	3.33	2.84	3.76

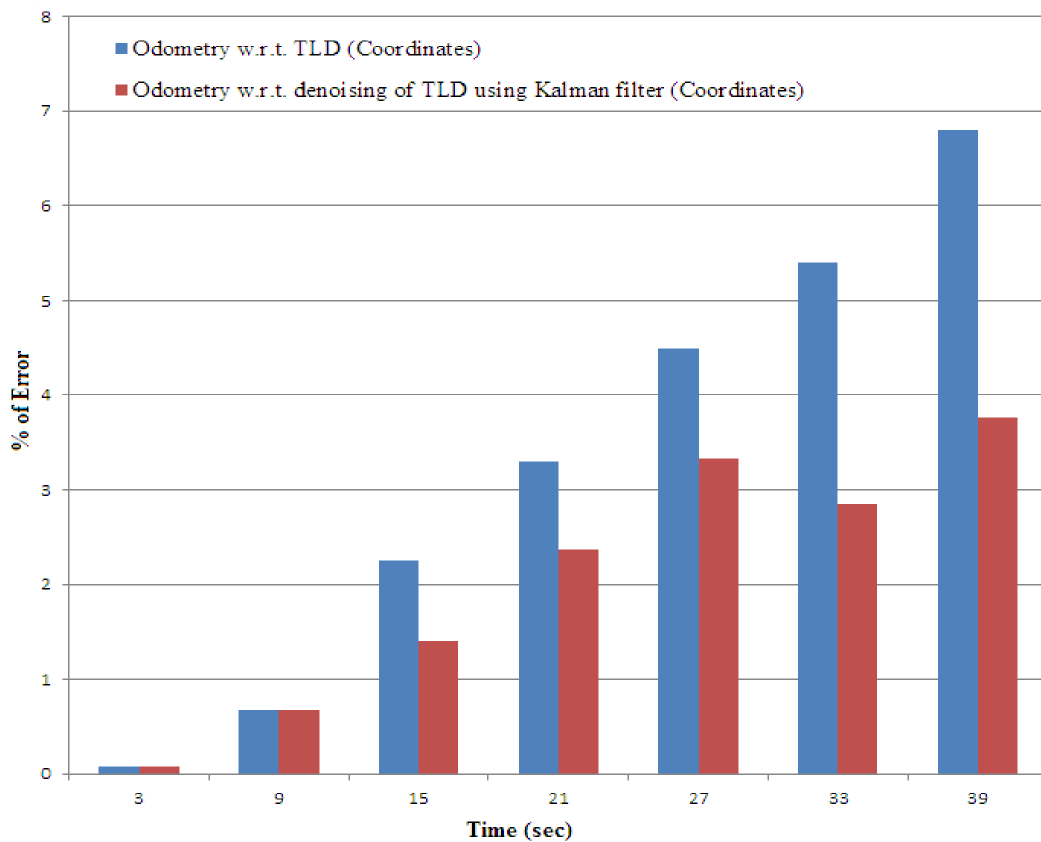


Figure 6.14: Percentage of error for TLD tracking and error after denoising using Kalman filter for test path 2

In the experiments, the deviations could be attributed to fast motion and illumination changes or fast rotation the mobile robot.

6.5 Conclusion

This work examines the application of safe path tracking problem of mobile robot navigation in an unknown environment among dynamic and static obstacles to create the shortest possible path from any current location to its target position. In this work, D* Lite algorithm with TLD tracking algorithm have presented for robot navigation in unknown environment. D* Lite algorithm starts search from the goal point towards the current position of the robot. D* Lite was implemented in real-time as a global planner plugin with the ROS Navigation Stack along with Dynamic Window Approach as the local planner. TLD tracking algorithm was used to track

the path of the mobile robot. The experimental and analytical results about D* Lite algorithm provide a fast re-planning methods in unstructured unknown environment. The percentage error between the obtained path with TLD tracking and Kalman filtering were computed to test the efficacy of the proposed algorithm. Here TLD algorithm generates error of the order 8-10% in test path 1 & 2 due to unsteady velocity and change in the direction of mobile robot. Kalman filter based denoising generates error less than 7% in test path 1 and 4% in test path 2 results with respect to TLD tracking.

In the future, this work is going to be optimized for local planning by the usage of other local planners like Timed Elastic Bands (TEB), Trajectory Rollout, Vector Field Histogram method etc. Since significant research has been done in computer vision, the object appearance could be described using different feature space such as (Sift, HoG, LBP etc.) that may result to more robust tracking.

The developed software architecture should be generalized to the extent that it can be used for other projects. The implementation should be extended to model behavioral aspect of mobile robot where reinforced learning based algorithm is used for environmental exploration.