TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SATHIS KUMAR THAMALA RAVIVARMA

DEMONSTRATION OF ULTRA WIDEBAND BASED POSITIONING FOR MECANUM WHEEL ROBOTS

Master of Science thesis

Examiner: Prof. Reza Ghabcheloo
Examiner and topic approved by the
Faculty Council of the Faculty of
Automation Science and Engineering
on 3rd January 2018

ABSTRACT

SATHIS KUMAR THAMALA RAVIVARMA: Demonstration of Ultra Wideband based Positioning for Mecanum Wheel Robots
Tampere University of Technology
Master of Science thesis, 56 pages
25th May 2018
Master's Degree Programme in Automation Engineering
Major: Factory Automation and Industrial Informatics
Keywords: Ultra wideband, UWB, IMU, gyro, Kalman filter, sensor fusion, path planner, path follower,marker detection

Ultra Wideband(UWB) communication have been increasingly used in indoor positioning systems to complement Global Positioning System(GPS) in indoor environments. This thesis demonstrates the accuracy and application capability of the technology in a small-scale factory environment. The factory environment consists of 3m by 3m area with designated loading areas for pallets.

The forklift type robots have forks to lift pallets and are fixed with mecanum wheels. The forklifts have to move the pallets based on the commands received through network effectively avoiding other robots and static obstacles in the area. The forklifts are 18cm by 25cm in size and have to move through the path of 30cm in width.

For positioning the robots in the environment, position from UWB is fused with onboard Inertial Measurement Unit(IMU) using extended Kalman filter. The path planner which plans from the current position of the robot to target area uses the map of the environment in OpenDRIVE format. A* planning is used to plan the path from the current position on the map to the goal. A dynamic obstacle grid is used to avoid moving robots in the vicinity of the robot.

Pure pursuit algorithm which selects a point on the path to follow is used for path following. Pallets are fixed with visual markers which are detected by the camera on forklifts. The marker detection provides the relative distance of pallets from the forklift which is used to pick and place the pallets. All these individual systems are integrated using state machines for seamless task execution. Four forklifts were effectively able to move pallets between loading areas without colliding.

PREFACE

This work has been accomplished as part of industrial thesis in Here Technologies, Tampere. I would like to thank the company especially Jari Syrjärinne for providing me this exciting opportunity. I would like to thank my instructor Jani Käppi for providing me the guidance and flexibility during the writing process.

I would like to thank my examiner Prof. Reza Ghabcheloo for his guidance during the thesis work and his patience throughout the thesis writing process.

My gratitude to all the people whom I have worked with during the thesis. This has been an interesting learning period for me not just technically but also in the writing process as well as in time and project management.

Tampere, 25.5.2018

Sathis Kumar Ravivarma

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

UWB	Ultra WideBand
IMU	Inertial Measurement Unit
GPS	Global Positioning System
TOA	Time of Arrival
TWR	Two Way Ranging
SDS-TWR	Symmetric Double Sided Two-Way Ranging
TDOA	Time Difference of Arrival
EKF	Extended Kalman Filter

1. INTRODUCTION

Indoor positioning systems are used to locate objects or humans inside closed environments such as buildings. There are wide range of applications for the indoor positioning systems from inventory management in factories, locating patients in hospitals, indoor navigation of unmanned vehicles to locating people in large shopping malls and offices to name a few.[1]

Global positioning system(GPS) is the most popular technology used for locating objects in the outdoor environment[1]. But there are certain limitations to GPS technology in indoor environments. The signals are weak when they reach the Earth's surface from GPS satellites to the receiver. They also suffer from multipath effect due to reflections in obstructed areas such as buildings[2]. They mostly need line of sight transmission between satellite and receivers. Achieving sub-metre level accuracy in indoor environment with satellite systems is infeasible and alternate systems have been considered[3].

Indoor positioning systems can be broadly classified into two types[1]. The first method is to use existing hardware infrastructure deployed for the purpose other than positioning such as Wi-Fi, Bluetooth and GSM. The second method requires extensive deployment of infrastructure and special hardware for positioning. RFID, infrared, ultrasound, lights, Ultra Wideband(UWB) are some of the technologies that fall under this category[4]. Further comparison and evaluation of these technologies can be found in but not limited to some of these surveys[3],[4],[1],[5],[6],[7]. UWB is one of the emerging technology for high accuracy indoor positioning[8].

UWB positioning systems use radio wave communication between transmitters and receivers over wide portion of the frequency spectrum to determine location. They relatively consume low power have become an important technology in high accuracy indoor positioning systems. UWB positioning systems have the capacity to provide high accuracy in the presence of objects due to its low interference with other radio waves and high-level of multipath resolution.[8] These advantages make it suitable for critical positioning applications and one such application is indoor navigation of robots.

The idea of the thesis is to implement UWB positioning in small-scale robots as an industrial project in HERE Technologies. These robots are used to demonstrate the accuracy of the positioning system designed by the company. The robot is built using 3D printed chassis and parts readily available in marketplace. Design process of the robot platform is out of scope of this thesis. Software for the robot is developed in modular architecture for easy testing and integration.

Though the hardware platform is small compared to real-world robots that achieve meaningful tasks, the robot serves as a portable test platform for the software which is designed to be scalable. Specifically, the robots are miniature forklifts which can lift small pallets. To our knowledge, there is no small size hardware platform available to create and test pallet picking algorithms with the software architecture of a full scale robot. Moreover, the modularity of the software and size of the forklifts allows testing scenarios using multiple robots which can be expensive in large scale. The small size of the forklifts also makes accurate localization challenging for pallet picking. The objective of the thesis is a demonstration of what the platform is able to achieve as a test platform and also the high accuracy of UWB based localization.

1.1 Objective

The objective of this thesis is to make small forklift robots to move autonomously using UWB positioning in the demonstration area. The demonstration area is 3 by 3 metres in size. Size of the forklifts is 15cm by 25cm. The forklifts should also detect and move pallets based on orders. Orders are communicated to forklifts through Wifi communication from visualization server functions of which are explained in chapter 3. The forklifts have cameras that detects pallets fixed with visual markers. These markers provide the relative distance of pallets from the robot using aruco marker detection library.

Demonstration area should be able to run multiple robots. Planning routes for forklifts have to be done using known map of the environment. The planner should also avoid other robots in the area and designated static obstacles. Since there are no other sensors for obstacle detection, static obstacles and robot position are transmitted over the network to other robots through visualization server.

A path follower should be implemented to follow the planned path. Pallet picking and placing algorithms are implemented independent of path planner. To achieve the demonstration objectives a state machine has to be designed. State machine manages when to do different tasks of the demonstration. All the systems have to be integrated with each other for execution of tasks autonomously with minimum intervention.

1.2 Thesis Structure and Contribution

My contribution to this demonstration is to implement kinematic modeling for the localization of forklifts using Extended Kalman Filter, path planning with obstacle avoidance, path following, parts of pallet picking and placing algorithms for forklifts and integration of these individual systems using state machine for task execution.

All the topics discussed in each chapter are huge field in itself, so the thesis structured in a way such that each chapter explains brief background of different methods and a detailed explanation of the chosen method along with experimental data and results.

Chapter 2 explains the theoretical background of different UWB positioning methods and the current positioning system. Chapter 3 describes the software and hardware design of the demonstration system. Motion and measurement models and its fusion with inertial measurement unit and UWB using extended Kalman filter is briefed in Chapter 4.

Obstacle tracking using occupancy grid and path planning using the obstacle and map information is explained in Chapter 5. Tracking of paths provided by path planner and state machine is explained in Chapter 6. Chapter 7 discusses the integration and communication of all modules and the experimental results of the demonstration.

2. BACKGROUND

2.1 Ultra Wideband

Any device which has the fractional bandwidth greater than 0.25 or occupying 1.5 GHz or more spectrum is defined as UWB device[9, pp. 12]. UWB have been used in military radar applications and communications for several years. When the Federal Communications Commission(FCC) allowed the unlicensed use of UWB communications in 2002, the technology received widespread interest.[10]

IEEE 802.15.4a standard targeted for UWB have been set up to create physical layer for low data rate communication with positioning capability. UWB radars differ from UWB sensor networks for various reasons. Sensor nodes work in harsh environment and cannot choose the environment whereas UWB radars can choose surroundings for minimum disturbance. Sensor nodes are affected by multiple interference whereas radars affect from narrowband jammers.[10]

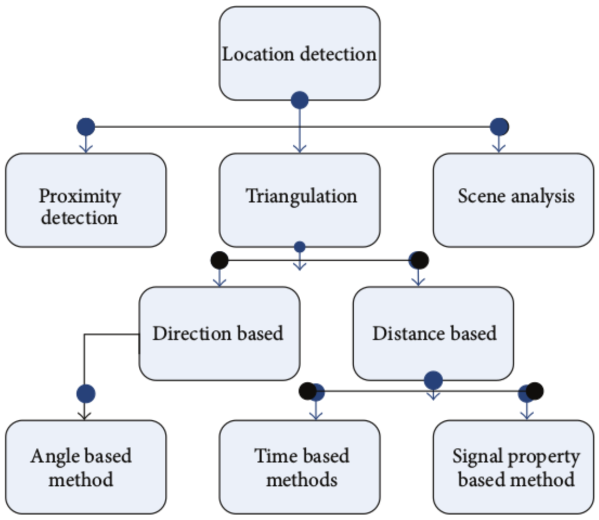


Figure 2.1 *Classification of Wireless Localization Techniques* [7, ,Fig.3]

Farid et al.[7, Ch.3] broadly classifies wireless indoor positioning systems as proximity, triangulation and scene analysis as shown in Figure 2.1. Proximity-based methods detects position by the location of beacon with the strongest signal as shown in Figure 2.2.a. Scene analysis methods are not related to UWB as they involve analysing large dataset for positioning.

Triangulation methods determine location by using geometric properties of triangles. Triangulation methods can be subdivided into Lateration and Angulation techniques. [7, Sec. 3.2]

Figure 2.2.c. shows angulation technique which determines target location by using the detected angle between the target and base station. UWB owing to its large bandwidth and multipath errors is prone to errors in determining angles[10].

Time based lateration techniques calculates position based on time propagation. There are different kinds of lateration techniques using measurement of propagation(TOA, TWR and TDOA). Lateration techniques determines location by measuring distance between different reference points as shown in Figure 2.2.b. [7, Sec. 3.2.2]

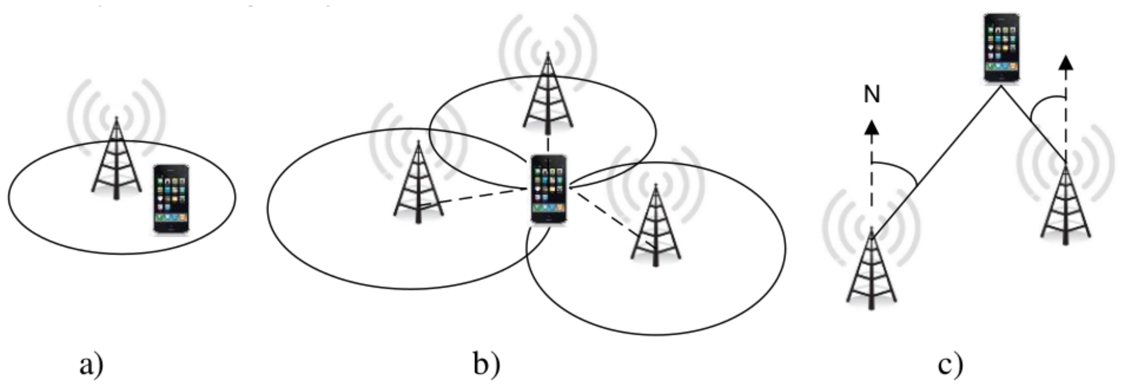


Figure* *2.2* *Localization Techniques a) proximity b) trilateration c) angulation [6, ,Fig.1]

Time based methods(lateration) are more suitable for UWB as they have better performance in multipath and indoor environments. They can achieve better accuracy by improving effective signal bandwidth. Large bandwidths of UWB signals allows highly accurate location estimates. [10]

2.1.1 Time of Arrival

To determine the location of a target node from a reference node, Time of Arrival(TOA) method determines the time taken by the UWB signal to travel from the reference node to target node. TOA method can also be called as One-Way Ranging(OWR) method. Since the speed of signal is constant, distance between reference and target node can be calculated by determining travel time. The target receiver node needs exact start time of signal from reference node, thus needing accurate synchronization of time between nodes.[7] Once the ranges of target node from multiple reference nodes at different locations are measured, the target location can be calculated by trilateration technique as shown in Figure 2.2.b. Small errors such as 3ns in synchronization between nodes can cause high degree of errors upto 1m[11].

2.1.2 Two Way Ranging

Time synchronization requirement in TOA method is eliminated to some extent in Two Way Ranging(TWR) method. Instead of using two clocks at two different nodes this method uses the clock on the transmitter side to measure the transmitting and arrival times[7]. Figure 2.3 shows two way ranging technique as described by Kwak et al.[11]. t_{round} represents the total time taken by the signal to reach receiver and return. t_{reply} is the time taken by the receiver to calculate and respond to the signal. Assuming no clock drift in the transmitter and receiver,the time of flight t_p can be measured by equation 2.1[11, Eq. 1].

$$t_p = \frac{1}{2}(t_{round} - t_{reply}) \tag{2.1}$$

TWR systems face significant delays if multiple range measurements have to be carried out consecutively. Clock crystals which calculate time can drift from reference time and is not unique for all devices. This induces an additional error known as clock drift. TWR method suffers from error measurements since clock drift is not considered in the calculation.

Let e_A and e_B be the crystal offsets of device A and B respectively, then the estimated time of flight is given by the equation 2.2[11, Eq. 2].

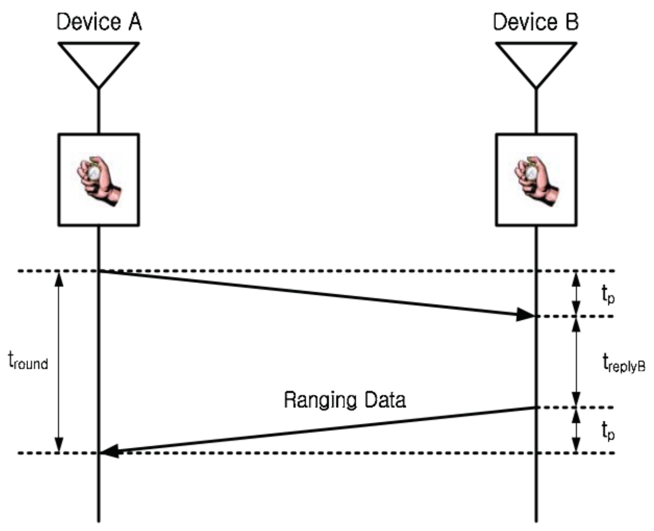


Figure 2.3 *Two Way Ranging*
[11, ,Fig.1]

$$\widehat{t_p} = \frac{1}{2}(t_{round}(1 + e_A) - t_{replyB}(1 + e_B)) \tag{2.2}$$

Error in the TWR method is the difference between true time of flight and estimated time of flight given by subtracting 2.1 from 2.2[11, Eq. 3].

$$\widehat{t_p} - t_p = t_p e_A + \frac{1}{2} t_{replyB}(e_A - e_B) \tag{2.3}$$

If t_p is small compared to t_{replyB}, then error due to clock drift is given by 2.4

$$\widehat{t_p} - t_p = \frac{1}{2} t_{replyB}(e_A - e_B) \tag{2.4}$$

2.1.3 Symmetric Double-Sided Two-Way Ranging

The method used in this thesis to determine UWB position is Symmetric Double-Sided Two way Ranging(SDS-TWR). SDS-TWR method minimizes the clock drift errors in TWR method. Figure 2.4 shows message exchange in SDS-TWR algorithm. This method calculates the range measurements twice symmetrically, once on each device and requires packets to be exchanged to four times. By following the notations in TWR method, theoretical time of flight in this method is given by equation 2.5 [11, ,Eq. 6]

$$t_p = \frac{1}{4}((t_{roundA} - t_{replyA}) + (t_{roundB} - t_{replyB})) \tag{2.5}$$

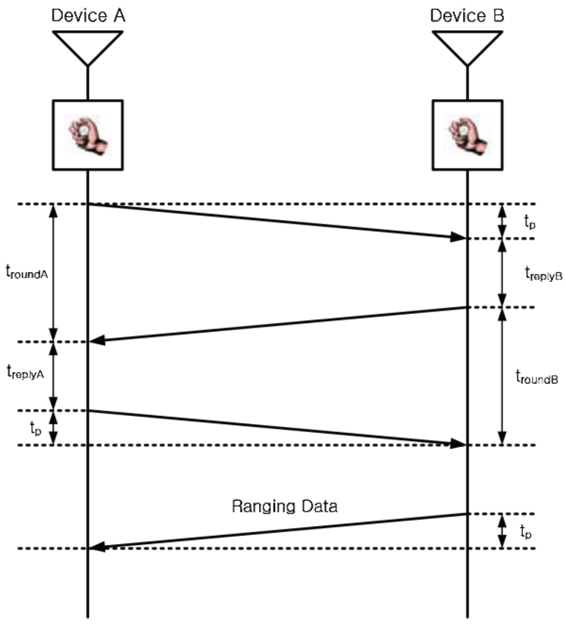


Figure 2.4 *Symmetric Double-Sided Two-Way Ranging* [11, ,Fig.2]

The estimated time of flight with clock drift e_A and e_B is given by equation 2.6 [11, ,Eq. 7].

$$\widehat{t_p} = \frac{1}{4}((t_{roundA} - t_{replyA})(1 + e_A) + (t_{roundB} - t_{replyB})(1 + e_B)) \tag{2.6}$$

Error in time of flight estimate in this method is given by equation 2.7 [11, ,Eq. 8].

$$\widehat{t_p} - t_p = \frac{1}{2}t_p(e_A + e_B) + \frac{1}{4}(t_{replyB} - t_{replyA})(e_A - e_B) \tag{2.7}$$

If t_{replyB} -t_{replyA} is chosen to be large compared to t_p, neglecting t_p term gives the ranging error due to clock drift as 2.8. [12]

$$\widehat{t_p} - t_p = \frac{1}{4}(t_{replyB} - t_{replyA})(e_A - e_B) \tag{2.8}$$

Compared with clock drift error in TWR method as given by equation 2.4, SDS-TWR method reduces the clock drift error by atleast half. [12]

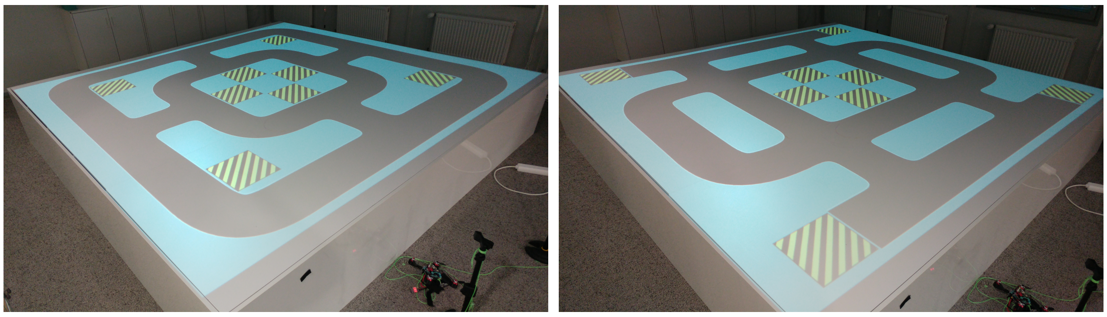
2.1.4 Time Difference of Arrival

Time difference of Arrival(TDOA) algorithm for UWB positioning systems measures the difference in arrival times from multiple transmitters at known location to a receiver at unknown location. The transmitters should have synchronized time between them. The difference in time from two transmitters to receiver gives a non linear hyperbolic equation. To determine the location of a receiver in 2D plane three time-synchronized transmitters are needed. For 3D location, four transmitters are needed. [13]

3. SYSTEM OVERVIEW

The complete system was developed in HERE Maps B.V, Tampere for public demonstration of UWB positioning. This chapter explains the system and platform which has been designed for demonstration by other team members in the company. My work on this thesis is to use this existing platform to achieve the demonstration objectives.

The demonstration area is 3m by 3m. The area does not have any fixed maps. Instead, the area can show different maps based on user selection. An image projector displays the selected map on a flat white table. Figure 3.1(a) and 3.1(b) displays different maps displayed on the demo table. The map projections are scaled to match the exact dimensions of the table.



(a) Demonstration Map 1 (b) Demonstration Map 2

Figure 3.1 *Different map images shown by the projector on the table*

The current map of the area is provided to forklifts in OpenDRIVE format. OpenDRIVE is an XML based map format for storing road networks and logic. Detailed explanation of this format is explained later in chapter 5. These different maps can be read by the forklifts at runtime when the map changes.

Architecture of the sytem is shown in Figure 3.2. UWB transmitter beacons are deployed around the demo table. These are fixed at known locations in the area. Movable UWB receiver beacons fixed in the forklifts provide positioning in the area.

Forklifts can read the location of all stationary beacons as configuration parameter for UWB positioning. The demo is limited to a maximum of four robots at a time.

The forklifts should move pallets between loading areas based on commands received from visualization server. Forklifts and visualization server are connected through Wi-Fi router. User inputs are available to provide orders from moving pallets from one place to the other. Users can also to add obstacles to the robots at run-time. The capability of the robots to replan routes when encountering static obstacles and other robots in the area have to be demonstrated.

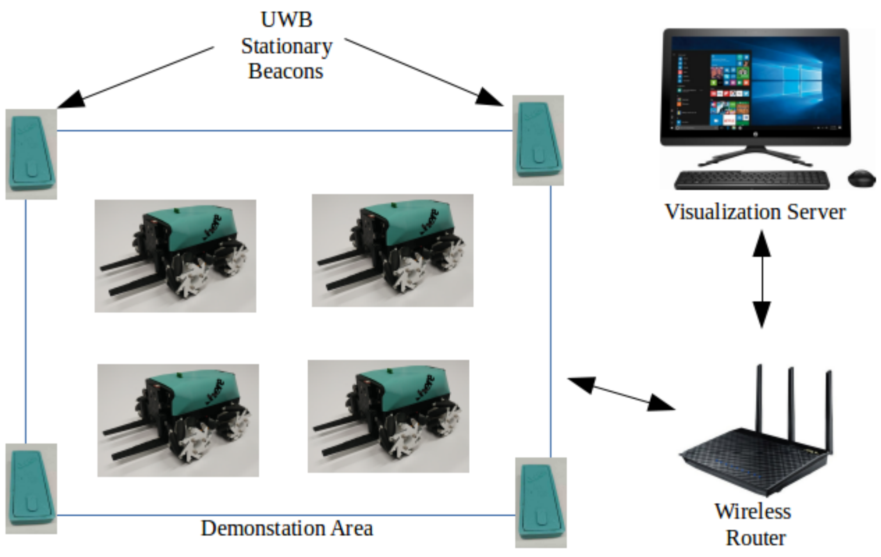
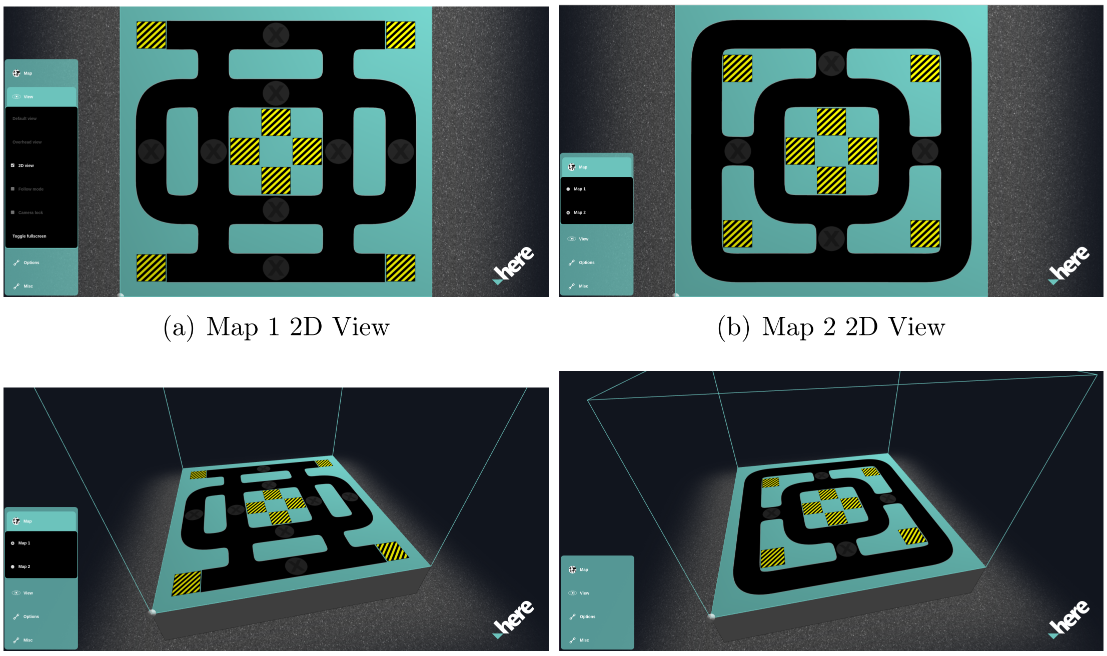


Figure *3.2 System Architecture*

Visualization server is a key part of the system which acts as a centralized command center also as a message router. Map to be used by each robot is assigned by the visualization server at the start and whenever it is changed by the user during runtime.

Each forklift broadcasts its own position and the planned path to the visualization server which forwards it to other robots connected to the server. Figure 3.3 shows different map visualization capabilities of the server. Visualization screen can display the planned path of all robots and their positions. User inputs of the system are directed from this system.

Designated static obstacles status of which are forwarded to robots whenever it is changed by the user and also when the robots start. The static obstacles are fixed



(a) Map 1 2D View (b) Map 2 2D View

(c) Map 1 3D View (d) Map 2 3D View

Figure *3.3 Different Maps available and their views in Visualization Server*

for each map and are displayed in the visualization. Users can enable the static obstacles by a single click.

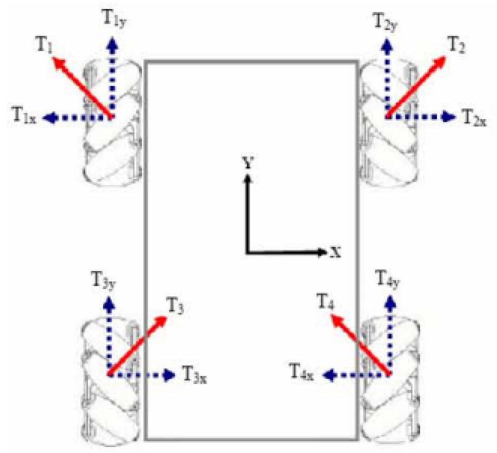
Order management system can assign orders automatically at random whenever there is a free forklift. Users can also provide orders manually by using touchscreen. Manual orders are disabled when there is no free forklift. Orders are sent to the forklifts based on ascending order of forklift id.

Order messages contain the location and direction of the loading area and their ids. Each message has two loading areas one for origin id to pick the pallet and the other for the target area for placing the pallet.

3.1 Robot Hardware

Forklifts consist of four mecanum wheels fixed to four servo motors. Mecanum wheel is designed by Bengt Ilon in 1973 for a Swedish company Mecanum AB. Compared to conventional wheels these have additional rollers at an angle around the circumference of the vehicle as shown in Figure 3.4(a). These rollers have their own axis of rotation additional to the rotation of the wheel. This feature translates a portion of the force in wheel direction to the direction normal to the rollers.

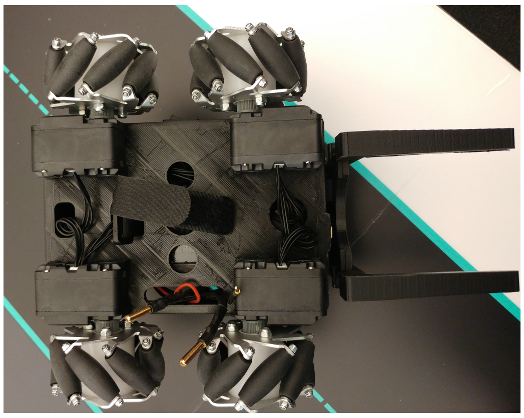
(a) Mecanum Wheel



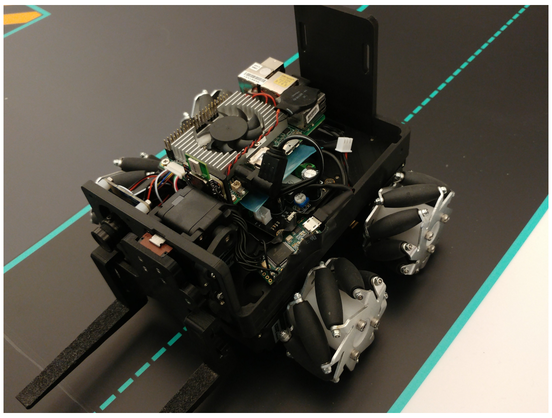
(b) Force vector of a typical Mecanum Wheel Robot[14]

Figure 3.4 *Mecanum Wheel Configuration*

These wheels are designed so that a robot can move in any direction without changing wheel direction. A typical four-wheel configuration and its force vector is shown in Figure 3.4(b). Let x and y denotes the direction of the robot in vertical and horizontal direction as shown in 3.4(b). Conventional wheels have force vectors only in y-direction. Mecanum wheels additionally have force vectors in x-direction.



(a) Servo mounted mecanum wheels



(b) Forklift hardware

Figure 3.5 *Hardware of the forklift*

Chassis of the forklift is built with 3D printing technology. It is powered by 7.4V lithium ion battery. Onboard processor board powered by Intel Atom processor runs on Ubuntu Linux. Forklifts also consists of MEMS 9-axis Inertial Measurement Unit(IMU), camera for marker detection and UWB receiver for positioning. Servo motor at the front provides lifting capacity for the forklifts.

3.2 Software System

The software system is a proprietary code written in c++. Software framework allows creation of modules that can run independently as a process inside the system and also allows sharing of data in the form of messages between these modules. This framework allows for dividing, developing and testing of complex tasks such as the pallet movement using forklifts into small individual components. Then these components are integrated into the system using the message passing capability of the software. Software modules created for the purpose of this demonstration is shown in figure 3.6.

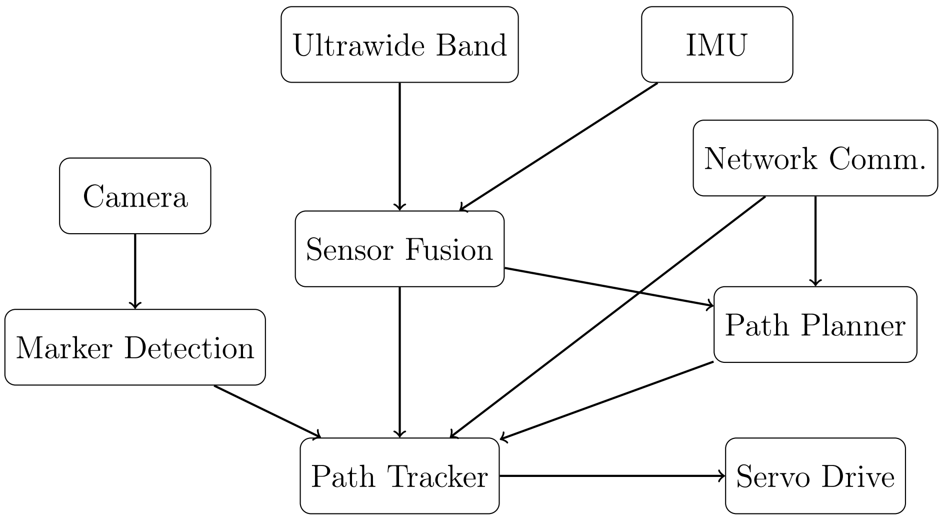


Figure 3.6 *Block diagram of the system*

Network Communication module is the only component that can communicate outside the software system. Position messages, planned path, current execution status of the forklifts and orders from visualization server are communicated through this module.

Ultrawide band module calculates location fixes from UWB range measurements using the UWB receiver fixed in the forklift.

Inertial Measurement Unit(IMU) module communicates with hardware IMU and sends rotational velocities in 3-axes(pitch, roll, yaw). **Servo Drive** module communicates commands to the motor.

Sensor Fusion module fuses measurements from IMU, UWB location and control values from path tracker to produce the pose of the robot in the map coordinate.

Camera module communicates with the onboard camera and forwards images. **Marker Detection** module detects markers on the pallet and provides relative distance from the forklift using aruco marker detection library.

Filter	**Output Variables**	**Update Rate(Hz)**
IMU	Rotation velocity	70
UWB	Position of UWB Antenna	20
Camera	Video	30
Sensor Fusion	Position and Orientation	70
Network Comm.	Position,path for all Robots	Variable(based on network speed)
Path Planner	Path points	Variable(based on order sent and obstacles)
Path Tracker	Linear and Angular Speed	70(30 for Visual Servoing during pallet pickup)

Table 3.1 *Update Rates of Different filters*

Path Planner module creates path from local maps from current location to target location avoiding obstacles.

Path Tracker module follows the path produced by the path planner module by sending commands to servo drive module. Added to this, the path tracker module manages the state machine for task execution.

Messages passed from one module in the system can trigger a process in another module. This method is used to create a sequence of events that are executed in the forklift. The execution rate of each module depends on the message that executes the process. The update rate of each of the modules in the system is shown in Table 3.1.

4. LOCALIZATION

This chapter describes the Localization method used in the forklifts. Localization is the method of estimating robot's location with respect to an external reference frame using sensors and map of the environment[15, pp.3]. Sensor measurements are not always accurate and are limited by noise, range and resolution. This creates uncertainty in the information about the robot location for autonomous execution of tasks. Hence robots ability to tolerate uncertainty is critical for mobile robot localization. [15, pp. 2]. Probabilistic method is one of the important approaches that

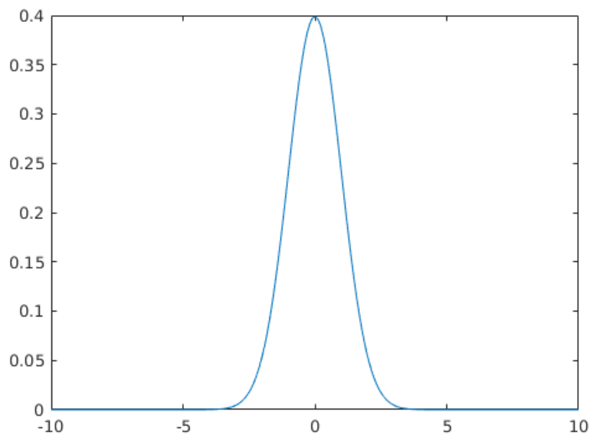


Figure 4.1 *Normal Distribution with mean μ=0 and variance σ^2=1*

considers the uncertainty in robot localization. It models the uncertainty in measurements as probability distributions.[15, pp.3] Gaussian filters models the uncertainty as multivariate normal distributions with mean μ and covaraince Σ. Normal distributions described with vectors instead of single scalar variable are called Multivariate Gaussian distribution[15, pp.11]. Figure 4.1 shows normal distribution of a scalar variable with mean zero and variance one. Further theoretical explanation on Gaussian filters can be found in Chapter 3 of [15].

Alternative to Gaussian filters are Non-parametric filters. Thrun et. al[15, pp 67] define Non-parametric filters as the filters that use approximate number of finite values as probability distribution instead of Gaussian function. Nonparametric filters are not considered for the localization in this thesis because of the popularity of Gaussian filters specifically Kalman filter in the industry.

Position and orientation of the body together are usually referred as pose. In general, a rigid body in three dimension system requires six variables(three cartesian coordinates and three angular orientations pitch, roll and yaw together called Euler angles) to determine accurate pose of the system. For robots in planar environments, the pose is determined by three variables (two cartesian coordinates and orientation of the robot called yaw).

In this thesis, the forklifts are always running on a flat surface. Hence, the localization deals with estimation of forklift pose in three variables {x, y, θ} where x and y are position of {B} with respect inertial frame {I} and θ denotes orientation($\theta > 0$ indicates anticlockwise direction) as shown in Figure 4.2

Extended Kalman Filter(EKF) is used for the estimation of forklift pose. Kalman Filter assumes the relation between the estimated variables known as state to be linear[15, pp. 35]. EKF is a non-linear form of Kalman Filter which linearizes the states using first order Taylor's expansion[15, pp. 49]. EKF is used because of the non-linear relationship between robot velocities and robot location.

Thrun et. al [15, pp. 19] describe two fundamental interaction between robot and external environment. Sensor measurement where external information about environment or robot is obtained. Control action which is applied to modify the robot state in the environment. This is the basis for EKF estimation.

EKF algorithm runs in two steps: Prediction and Update. In prediction step, information about the robot motion is used to predict the state and covariance. In the update step, measurement data is integrated into the estimation. The update step decreases the uncertainty of the robot pose whereas the prediction step increases the uncertainty of the robot pose[15, pp. 39].

Equations 4.1- 4.5 show the equations of EKF[15, pp.51].

Prediction Stage:

$$\mu_{t|t-1} = g(u_t, \mu_{t-1|t-1}) \tag{4.1}$$

$$\Sigma_{t|t-1} = G_t \Sigma_{t-1|t-1} G_t^T + R_t \tag{4.2}$$

Update Stage:

$$K_t = \Sigma_{t|t-1} H_t^T (H_t \Sigma_{t|t-1} H_t^T + Q_t)^{-1} \tag{4.3}$$

$$\mu_{t|t} = \mu_{t|t-1} + K_t(z_t - h(\mu_{t|t-1})) \tag{4.4}$$

$$\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1} \tag{4.5}$$

Notations used in EKF equations:

$\{x_t\}$ State of the Robot at time t given the state.

$\{z_t\}$ Measurement e at time t.

$\{\hat{z}_t\}$ Measurement e at time t.

$\{u_t\}$ Control data at time t.

$\{\mu_{t|t-1}\}$ Estimate of mean of the state at time t given μ at t-1.

$\{\Sigma_{t|t-1}\}$ Covariance of the state at time t given Σ at t-1.

$\{G_t\}$ State Transition Matrix at time t.

$\{R_t\}$ Transition Noise Covariance at time t.

$\{H_t\}$ Measurement matrix at time t.

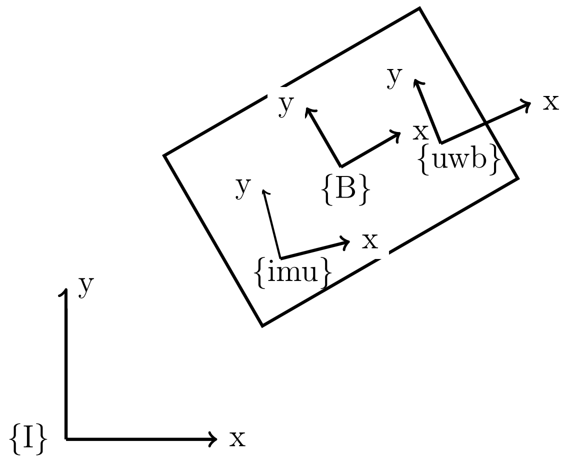
$\{Q_t\}$ Measurement Noise Covariance at time t.

$\{K_t\}$ Kalman Gain at time t.

$\{I_n\}$ Identity matrix of size n.

Added to these equations, EKF needs measurement and motion model. These models are used to convert the sensor measurements and control action of the robots into EKF equations.[15, pp.91]

UWB and IMU are the sensors present in the forklifts. UWB location which can be classified as absolute localization is based on external reference and can provide accurate measurements over time but cannot provide measurements always[16]. IMU measurements are internal sensors that provide continuous measurements in high frequency but drifts with time. Hence only UWB measurement is used in the Update step of EKF. IMU measurement is used in prediction step in the motion model.



[h!]

***Figure** 4.2 Coordinate system of Sensors*

Notations used for Motion model:

{x, y, θ} - Pose of the body frame {B} with respect to {I}.

{x_{uwb},y_{uwb}} - Position measurement of {UWB} frame with respect to {I} from UWB module.

{$\omega_1, \omega_2, \omega_3, \omega_4$} - Rotational velocities of individual wheels marked as shown in 6.1

r - Radius of the wheel.

{lx, ly} - Half of the distance between the center of the wheels in x and y direction as shown in Figure 6.1.

{v_x, v_y} - Instantaneous Linear velocities in the x and y direction of body frame {B} at {B}.

{$v_{x|t}$, $v_{y|t}$} - Linear velocities in the x and y direction of body frame {B} at time t.

{V_x, V_y} - Calculated linear velocities in the x and y direction of body frame {B} from control values.

{ω} - Angular velocity of the forklift about an axis perpendicular the body frame {B}.

$\{\omega_C\}$ - Calculated angular velocity of the forklift about an axis perpendicular the body frame {B} from control values.

Figure 4.2 shows different coordinates systems used by the forklifts. The center of the forklifts is represented by body frame {B}. Inertial coordinate {I} is the stationary reference frame in which all the forklift locations are measured. Localization of the forklift in this context is finding the pose of the coordinate system {B} with respect to the inertial coordinate {I}. UWB module provides the location measurement of the UWB sensor at {UWB} with respect to {I}. {IMU} is the location of IMU with in the forklift.

Added to the pose, linear velocities(v$_x$ and v$_y$) and angular velocity(ω) of the forklift are estimated by EKF. All these variables that affect the localization of the robot together are called state.

As explained before, the state of the forklift to be determined by EKF is given by equation 4.6.

$$x_t = \begin{bmatrix} x & y & \theta & v_x & v_y & \omega \end{bmatrix}^T \tag{4.6}$$

Motion Model:

Motion model propagates the state vector with time about the robot movement. Equation 4.7 shows the Gaussian motion model with error R_t.

$$x_{t|t-1} = g(u_t, \mu_{t-1|t-1}) + R_t \tag{4.7}$$

In the IMU measurements only gyro measurement which is the rotational velocity ω_{imu} in the direction perpendicular to the forklift is used for propagation of motion. Linear velocities are modelled as errors. Equation 4.8 shows the motion model used for propagation in prediction calculated from the kinematics of the mecanum wheel robot.

$$g(u_t, \mu_{t-1|t-1}) = \begin{bmatrix} x_{t-1} + v_{x|t-1} \times dt.cos\theta_{t-1} + v_{y|t-1}.dt.cos(\theta_{t-1} + \pi/2) \\ y_{t-1} + v_{x|t-1} \times dt.sin\theta_{t-1} + v_{y|t-1}.dt.sin(\theta_{t-1} + \pi/2) \\ \theta_{t-1} + \omega_{imu}.dt \\ v_{x|t-1} \\ v_{y|t-1} \\ \omega_{t-1} \end{bmatrix} \tag{4.8}$$

Measurement model

Measurement values z_t modeled with Gaussian noise Q_t is given by 4.9.

$$z_t = z_o + Q_t \tag{4.9}$$

where z_o is the true value of the measurement. Measurement values of position from UWB sensor is given by 4.10.

$$z_t =^I P_{UWB} = \begin{bmatrix} x_{uwb} \\ y_{uwb} \end{bmatrix} \tag{4.10}$$

Given the state heading at x$_{t|t-1}$ as θ_t. Then measurement estimate of {UWB} with respect to {I} is given by,

$$^I\hat{P}_{UWB} =^I \hat{R}_B \times^B P_{UWB} +^I \hat{P}_B \tag{4.11}$$

where the rotation matrix $^I\hat{R}_{UWB}$ can be calculated using robot orientation estimate θ_t as given by equation 4.12. $^{UWB}P_B$ is measured from the 3D diagram of the forklift chasis.

$$^I\hat{R}_{UWB} = \begin{bmatrix} cos\theta_t & -sin\theta_t \\ sin\theta_t & cos\theta_t \end{bmatrix} \qquad ^BP_{UWB} = \begin{bmatrix} a \\ b \end{bmatrix} \qquad ^I\hat{P}_B = \begin{bmatrix} x_t \\ y_t \end{bmatrix} \tag{4.12}$$

where a=0.05 and b=0.0 are the values measured from the 3D diagram. Estimate of the measurement at time t by applying equations 4.11 and 4.12 is given by 4.13.

$$\hat{z_t} = h(\mu_{t|t-1}) = \begin{bmatrix} a.cos\theta_t - b.sin\theta_t + x_t \\ a.sin\theta_t + b.cos\theta + y_t \end{bmatrix} \tag{4.13}$$

4.1 Sensor Fusion

Sensor Data Fusion is the process of combining incomplete and imperfect pieces of mutually complementary sensor information in such a way that a better understanding of an underlying real-world phenomenon is achieved[17, pp. ix]. In the context of this thesis, it is the process combining UWB and IMU measurements from path tracker for localization of forklifts.

Algorithm used in the sensor fusion module is shown in Figure 4.3. The algorithm waits for first valid measurement from UWB for initialization.

```
1: if $UWB\ Valid$ then
2:     Initialize State Vector $\mu$ and Covariance Matrix $\Sigma$
3: end if
4: while $\mu$ initialized do
5:     if $IMU$ Received then
6:         Predict EKF
7:         Send Position
8:     end if
9:     if $UWB$ Received then
10:        Update EKF
11:    end if
12: end while
```

Figure 4.3 *Algorithm of Sensor Fusion Filter*

Initialization of State:

Initial heading of the forklift is not known and assumed to be zero. Forklifts are always started at zero heading in the map. First valid UWB measurement is used for initialization of x and y values. By applying θ_t=0 and measurement values x_{uwb} and y_{uwb} to 4.13, initial state vector is given by,

$$\mu_0 = \begin{bmatrix} x_{uwb} & y_{uwb} & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{4.14}$$

Forklifts are always assumed to be stationary at the start, so all the velocity values of the state vector are assumed to be zero.

Prediction Equations:

State transition matrix is the Jacobian of the state function 4.8 which is given by.

$G_t = \frac{\partial g}{\partial x_t} =$

$$\begin{bmatrix} 1 & 0 & -dt*(sin\theta*v_x + sin(\theta+\pi/2)*v_y) & cos\theta*dt & cos(\theta+\pi/2)*dt & 0 \\ 0 & 1 & dt*(cos\theta*v_x + cos(\theta+\pi/2)*v_y) & sin\theta*dt & sin(\theta+\pi/2)*dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

R$_t$ is the process noise covariance which is the model of the noise in the prediction process. These values are obtained experimentally during testing based on the behaviour of sensors.

$$R_t = \begin{bmatrix} 10^{-5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-8} & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-6} \end{bmatrix} \tag{4.16}$$

When the IMU measurements are received, it is used in the motion model for EKF prediction. Steps 8-12 in the algorithm describes the prediction stage. Position measurements are sent to other modules in this stage.

Update Equations:

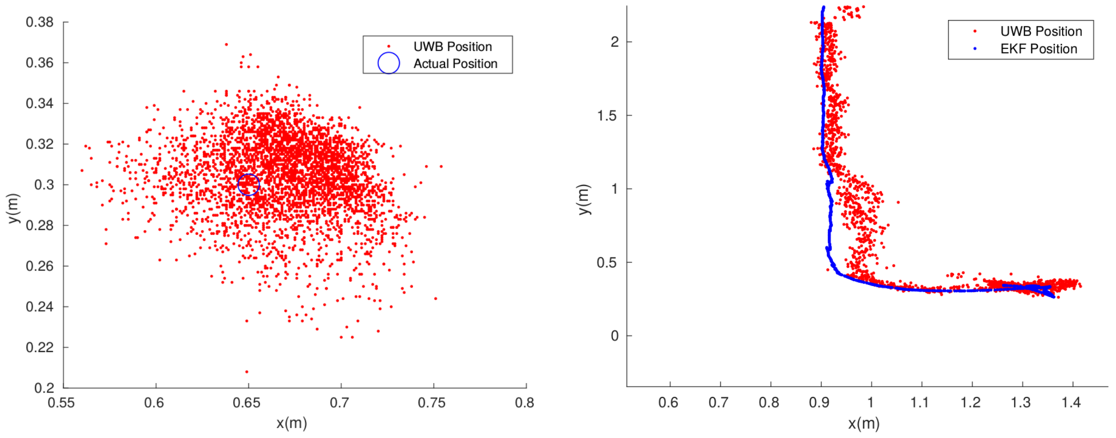
When UWB measurement is received, the measurement vector $\hat{z}_t$ is obtained using equation 4.13. Measurement matrix is found by the Jacobian of measurement function given by 4.17.

$H_t = \frac{\partial h}{\partial x_t} =$

$$\begin{bmatrix} 1 & 0 & -a.sin\theta - b.cos\theta & 0 & 0 & 0 \\ 0 & 1 & a.cos\theta - b.sin\theta & 0 & 0 & 0 \end{bmatrix} \tag{4.17}$$

Measurement noise covariances are modelled after the behaviour of sensor measurements. Figure 4.4(a) shows the UWB raw position when the forklift is stationary. Considering the movement and accounting for delays in position input covariances for measurement noise used for the forklift sensors are given by 4.18

$$Q_k = \begin{bmatrix} 0.35^2 & 0 \\ 0 & 0.35^2 \end{bmatrix} \tag{4.18}$$



(a) UWB raw data at stationary position (b) Position estimate of EKF compared to UWB measurement

Figure 4.4 *Comparison of UWB measurement data and Sensor Fusion Estimate*

Figure 4.4(b) shows the comparison of UWB raw data to EKF estimate. It can be observed that the solution is smooth and within the UWB measurement in straight lines. The actual position of the UWB is within the UWB solution, thus when turning in curves the solution deviates from the UWB position because of delay in velocity propagation. Having wheel speed sensors could propagate the solution faster towards the actual position. For the demonstration, this deviation is within the tolerance.

5. PATH PLANNER

This chapter explains the path planner algorithm used in the forklifts. Planning in robotics can be defined as the algorithm to convert high-level tasks given to robots into low-level achievable instruction for the robots to execute[18, pp. 3]. The overall objectives of the path planner in this thesis is to provide a collision-free plan to the target for the forklifts which satisfies the below constraints.

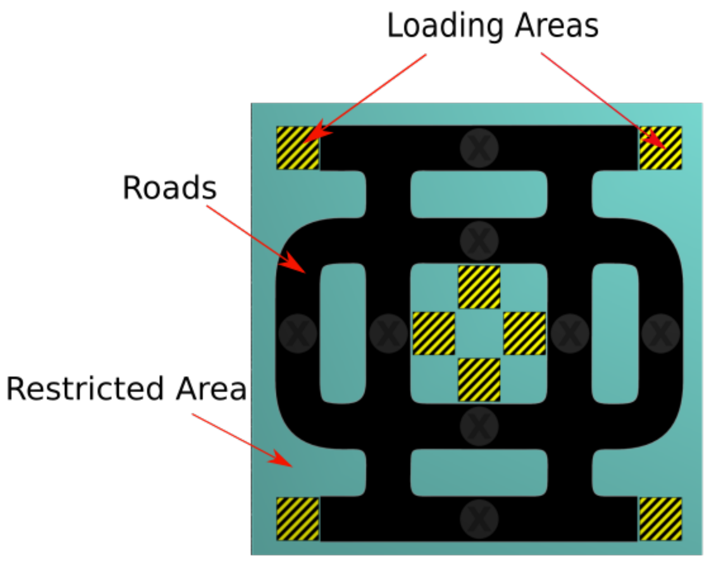


Figure 5.1 *Constraints for Planner:Roads and Loading Areas*

The constraints for the path planner which are derived from the overall objectives of the demonstration are described below.

- Map of the area is provided as roads and the planned path should always follow the roads.
- Always avoid static obstacles. Static obstacles which can be added and removed by the user. These are considered permanent obstacles and must be avoided immediately.

- Avoid collision with dynamic obstacles. Location of other forklifts is provided through network communication. These are always considered as moving (dynamic) obstacles.
- Loading areas are always specified outside the roads but near the edge of the roads. Pallet picking and placing is implemented by path tracker and is outside the scope of the planner.

Figure 5.1 shows the loading areas, roads and restricted areas of one of the map.

The problem is split into two parts: 1) To create a global plan based on the map and static obstacles. 2) To track the moving(dynamic) obstacles and create a local plan to achieve the global plan or change the global plan if needed. The collision-free local plan is used by the path tracker module for forklift movement.

Global planning is implemented using graph search(discrete planning) algorithm and maps. Maps for creating the global plan is provided in OpenDRIVE format. OpenDRIVE format provides the map data as roads. Section 5.1 explains the OpenDRIVE format and conversion of map data into state graph for planning. A graph contains a list of vertices and edges which are connection between vertices.

Using the state graph, a graph search algorithm is implemented in section 5.2 to create a global plan without any obstacle cost. For adding obstacle cost to graph search, obstacles have to be tracked. This is implemented using an occupancy grid. Occupancy grids discretize the map area and store the occupancy as probability values to account for uncertainty in sensors. This type of grid is used to have the flexibility of adding range sensors in the future.

Section 5.3 explains the obstacle tracking, addition of obstacle cost to global planning and local planning with dynamic obstacles. Static and dynamic obstacles are tracked in different occupancy grids. Obstacle cost for global planning is implemented using the cost from the static grid.

Separate occupancy grid for static and dynamic obstacles reduces the replanning frequency in global planning as the static map does not change frequently. This also gives the flexibility to change a global plan by adding an obstacle to the static grid if the local plan is infeasible. Local planning handles the dynamic obstacles and requests for a new global plan by adding and removing temporary static obstacles to global planner.

Problem Formulation for graph search

- Let x $\in$ X be the state which is the distinct location of a point on the map. X is the state space containing the set of all possible states.
- Let U(x) $\subset$ X be the list of neighbour states x$'$ $\in$ U(x) of state x where x$'$ $\neq$ x [18, pp. 28]. This state graph X along with the neighbour connections is state graph.
- Planning algorithm must find the path as a list of states from start state x_S to goal state x_G with minimum cost to the goal. Cost for the forklifts is to minimize the travel time. Assuming constant linear and angular velocity of forklifts, using distance and turning as cost is reasonable.
- Planning algorithm must avoid static obstacles and replan immediately when encountered one irrespective of the distance from current location.
- Assuming the static obstacle can be removed, the plan should stop before the static obstacle if no collision-free paths are available. This reduces the travel time when the static obstacle is removed.

Background - Graph Search

Discrete planning or graph search methods search the state space systematically for all states until a solution is reached or returns if there is no solution. If the state space is infinite, these methods tend to run infinitely. Thus a basic requirement for this method is to a have finite or countable number of states. [18, pp. 28]

There are three broad classifications based on search direction in discrete planning methods. Forward search, Backward search and Bidirectional search. [18, pp. 28]Forward search method searches the state space from start state and the search is stopped when the goal is reached or returns when no solution is found after exploring all the states. Backward search methods searches the state space from goal to start. Bi-directional search as the name indicates searches the state space in both directions until the two searches meet.[18, pp. 32]

Bi-directional searches are more complicated than forward and backward search algorithms but provide faster search results. For a simple planning problem of this thesis with finite states, one directional methods are sufficient. Forward search algorithms can be applied to backward search algorithms by starting the search from goal state. Hence, forward search algorithm is considered for this planning problem and the backward search does not provide any advantage than the former and vice versa.[18, pp. 39]

There are two main schemes in the general search algorithms which provides faster convergence to the solution if one exists. First one is Dijkstra's and A*(A-Star) which maintains a list of visited and unvisited states and searches through the lists to find the solution. The other one is the value iteration method which finds optimal solution from every state to the goal. There are other search schemes such as Breadth-first, Depth-first etc., but only the two schemes are considered for their popularity in path planning.[18, pp. 35]

Dijkstra's forward search algorithm works by categorizing each state in the state space into three kinds during the search. 1) Unvisited states that are not visited yet. 2) Dead States that have been visited and each of the next possible states are also visited. 3) Alive States that are visited but atleast one of the next states is not visited. At the start of the search, start state is the only alive state. [18, pp. 37]

The alive states are stored in a priority queue and the next state to be explored from the queue is provided based on the priority function. The algorithm runs until the goal state is found or the priority queue is empty i.e. all states are explored and no path is found. The kind of priority function used determines the efficiency of the search method.

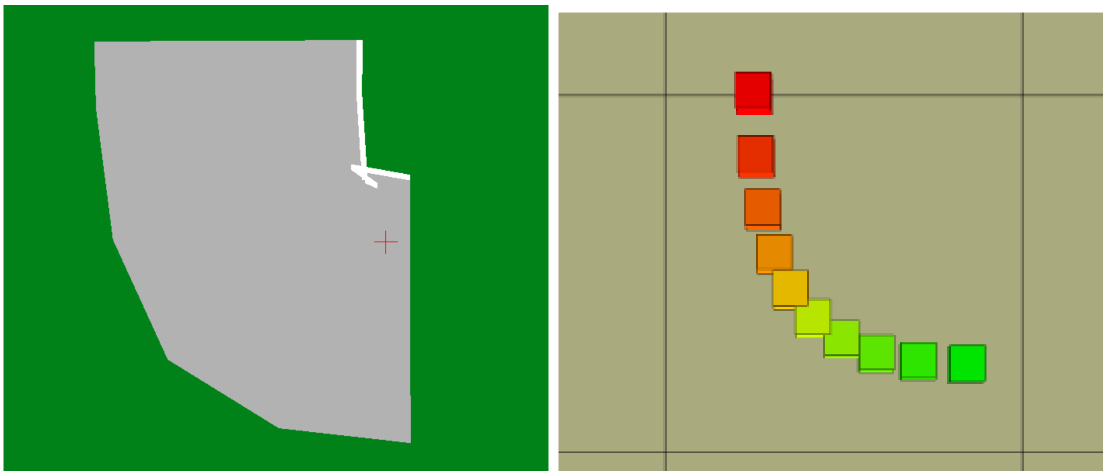
The priority function calculates the cost-to-come from the start state to the current state. State with the lowest cost-to-come is given highest priority. Thus in the priority queue, states with the lowest cost-to-come are first explored. All the states are explored systematically by minimizing cost until the goal is found or return if there is no solution. One necessary requirement is for the cost function to be non-negative to avoid revisiting dead states(creates infinite loops).

A* algorithm works similar to the Dijkstra's, except in the calculation of cost for the priority queue. In addition to the cost from start to current state another heuristic(assistance) cost which provides an underestimate of the cost to goal is added. This additional heuristic cost directs the search towards the goal faster than Dijkstra's by assigning lower cost to states closer to the goal. Thus A* is more efficient than Dijkstra's for most cases and preferable because of faster computation times.

Value iteration method iteratively computes the optimal cost to the goal from every state. This method is similar to Dijkstra's algorithm in finding the solution except that it does not maintain any lists. Hence reducing complexities for large state spaces. For this thesis, with the small number of states, A* is used over value iteration as there is no need for solution from every state and A* can provide an optimal solution from start to goal.[18, pp. 55]

5.1 OpenDRIVE Maps

The State graph X is extracted from the OpenDRIVE maps. This format is used because of the popularity of its use in automotive companies[19]. OpenDRIVE format is an xml based syntax which divides the map into a number of road sections. OpenDRIVE map can contain any number of road sections but should have atleast one road section[20].



(a) Road Section (b) Points extracted from Road Section

***Figure* 5.2** *Sample OpenDRIVE Map with one road*

Each road section has the information about the geometry and profile of the road, number of lanes in the road, connections to other roads etc. Discussing all the available options of OpenDRIVE format is out of scope of this thesis and only the XML tags that are used are discussed further. Detailed specification of OpenDRIVE format can be found in [20].

A sample OpenDRIVE map with one road is shown in Program 5.1. Image of the road and the map points extracted from it are shown in 5.2(a) and 5.2(b) respectively.

Program 5.1 *Sample OpenDRIVE Road Element*

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<OpenDRIVE>
  <header east="88.068139276165681" north="
      75.454771912232701" revMajor="1" revMinor="4" south="
      -2.6450232105176767" west="0">
  </header>
  <road id="98" junction="86" length="9.0104831542520678"
      name="_along">
    <link>
      <predecessor contactPoint="start" elementId="30"
          elementType="road"/>
      <successor contactPoint="start" elementId="57"
          elementType="road"/>
    </link>
    <planView>
      <geometry hdg="-3.1390204013397978" length="
          9.0104831542520678" s="0" x="30.754649345320242" y=
          "25.977241937401093">
        <paramPoly3 aU="0" aV="0" bU="10.660850785432578" bV
            ="0" cU="-7.097389422503575" cV="
            -4.5493778325431746" dU="1.1878204543096822" dV="
            -1.5164592775143679"/>
      </geometry>
    </planView>
    <elevationProfile>
      <elevation a="0" b="0" c="0" d="0" s="0"/>
    </elevationProfile>
    <lanes>
      <laneSection s="0">
        <center>
          <lane id="0" level="false" type="none">
            <link/>
            <roadMark laneChange="both" sOffset="0" type="none" weight="standard" width="0"/>
          </lane>
        </center>
        <right>
          <lane id="-1" level="false" type="driving">
            <link>
              <predecessor id="1"/>
              <successor id="-1"/>
            </link>
            <roadMark color="standard" laneChange="none" sOffset="0" type="solid" weight="standard" width="0.1016"/>
            <userData>
              <referenceLaneBoundary index="0"/>
            </userData>
          </lane>
        </right>
      </laneSection>
    </lanes>
  </road>
</OpenDRIVE>
```

An XML parser strips data from the OpenDRIVE file. Attribute values from the XML file provides the information about the roads.

Road sections provides the profile of the path as parametric cubic polynomial with parameter p in the range[0;1][20, pp. 46]. Figure 5.2(b) shows a road created with 10 points(p=0,0.11,0.22,..0.88,1). By choosing the right interval, any number of finite points can be created on the path of the road. Since the OpenDRIVE maps available for this thesis had small road sections in uniform size. Same number of points have been created from each road. This need not be the case for all maps and can be varying for each road based on the length of the road section.

To proceed further all the states x and the list of neighbour states U(x) for each state must be extracted from the OpenDRIVE map. Algorithm used for extracting states and connections is shown in Figure 5.3. Steps 1-6 in algorithm 5.3 extracts

```
 1: for road do
 2:     From <road> Get roadId, junctionId
 3:     From <link> Get Successor and Predecessor Id, contactPoint
 4:     From <geometry> Get x, y, heading
 5:     From <paramPoly3> Get aU, aV, bU, bV, cU, cV, dU, dV
 6: end for
 7: for road do
 8:     Find points in local coordinate $u_{local}$ and $v_{local}$
 9:     Transform points to global $x_{global}$ and $y_{global}$
10:     Create Unique id i
11:     Find neighbour states $x'_i$
12:     Add $x_{global}$, $y_{global}$ and $x'_i$ to State Space to X
13: end for
```

Figure 5.3 *Algorithm for Creating Map Points*

information about each road. Profile of the road is specified by parametric cubic polynomial in a local frame. Steps 7 and 8 in algorithm 5.3 uses the extracted polynomial parameters to create map points in global frame. Global frame is the frame at the origin of the map. Equations used for extracting map points in local frame from polynomial is given by 5.1 and 5.2. p is the parameter value between 0 and 1 where p=0 gives the location of first point and p=1 provides the location of last point in the local frame[20, pp. 46].

$$u_{local} = a_U + b_U.p + c_U.p^2 + d_U.p^3 \tag{5.1}$$

$$v_{local} = a_V + b_V.p + c_V.p^2 + d_V.p^3 \tag{5.2}$$

Translation and rotation of local frame is provided by the parameters x, y and heading. Location of each point in the global frame is found using rotation and translation given by equations 5.3 and 5.4.

$$x_{global} = u_{local}.cos(heading) - v_{local}.sin(heading) + x \tag{5.3}$$

$$y_{global} = u_{local}.sin(heading) + v_{local}.cos(heading) + y \tag{5.4}$$

Any number of unique points can be created by using a unique parameter value p between 0 and 1. Each point can have a unique number m=1:M on the road where M is the number of points for each road. For the small maps in the thesis, this number is chosen to be M=10. This along with a unique road id provided in the map is used to create a unique global id i where i=1,2..N and N=N_R*M(N_R is the total number of roads). This unique id is assigned to each map point as shown in step 10 of algorithm 5.3 for identifying the next possible states from current state. These identifiers are also required to retrace the path once the goal is found which is explained in section 5.2.

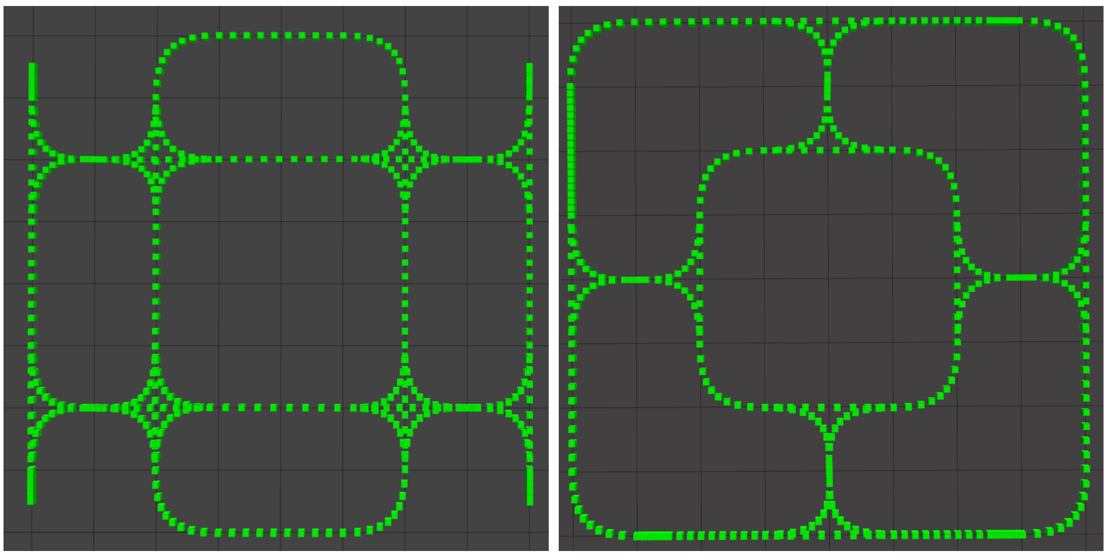

(a) Points from map 1 (b) Points from map 2

Figure 5.4 *Map Points from different OpenDRIVE Maps*

To find the connection between map points, connection information from roads is used. Each road has a unique road id . Links between roads are specified by successor and predecessor road id. The links are road ids of other roads if connection is not ambiguous. If a road has multiple successor or predecessor then a junction id is used. Junction are nothing but a group of roads that connect different roads. An example of a group of connecting roads in a junction is shown in Figure 5.5. These group of roads together are provided by a unique junction id in OpenDrive maps. This is used to extract multiple road connections of the road if exists.[20]

Possible next states of the current state are found based on successor and predecessor road id provided in the road information for last points (p=0 or 1). For other points in each road(p $\neq$ 0 and p $\neq$ 1), the previous or next point in the same road are the possible next states. Figure 5.4 shows the map points extracted from the OpenDRIVE maps. With this information the neighbour states U(x) of all x is determined.

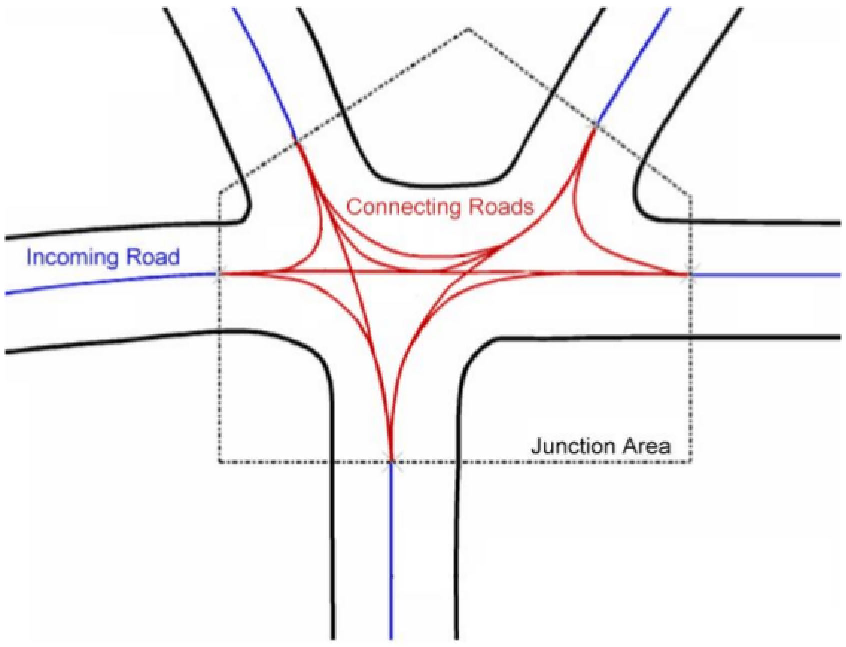


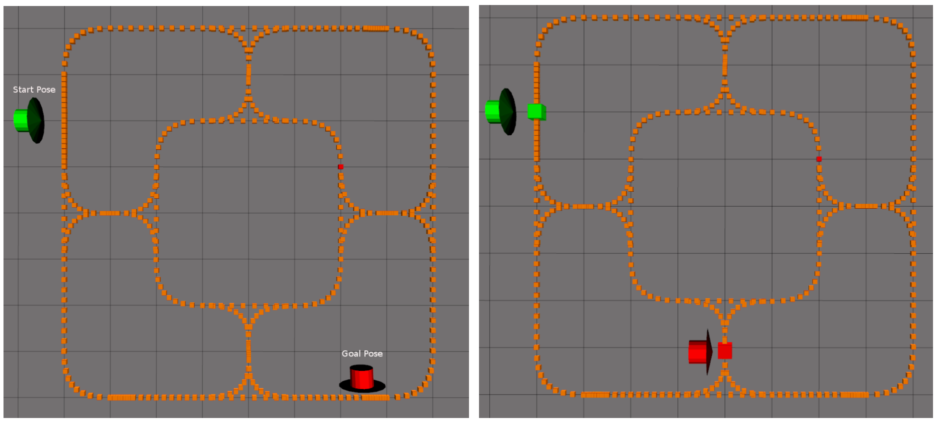
Figure 5.5 *Example: Connecting Roads(in Red) of a Junction*

5.2 Graph Planning

The next step is to implement a planning algorithm from start to goal pose using the state graph X extracted from map in the previous section. Figure 5.6(a) shows a sample start and goal pose for which the planning has to be implemented.

Given any random start pose and goal pose within the map area, A* planner must find the path using the map points. Since the state space is a set of discrete locations, a suitable state has to be selected from the state space for each pose. State with the closest Euclidean distance is used as matching state for each pose. By this method a start state $\mathrm{x}_S \in \mathrm{X}$ and a goal state $\mathrm{x}_G \in \mathrm{X}$ is chosen for planning. Figure 5.6(b) shows an arbitrary start and goal pose along with their chosen states.

Figure 5.7 shows the A* algorithm used for map planning. Given the start state x_S and goal state x_G in the state space, this algorithm finds the list of states from start to goal. Using the goal state a heuristic cost $\mathrm{H}(\mathrm{x},x_G)$ is calculated from every state to the goal state. This cost must be an underestimate i.e. less than the actual cost from the a state to goal state to provide an optimal path. Euclidean distance is the shortest possible distance between two points in a 2D plane. Hence this distance is used as the heuristic cost.



(a) Random start(Green arrow) and goal(Red arrow)
(b) Selected state from the map(state space)

Figure 5.6 *Selection of state from state space based on Euclidean distance*

Figure 5.8 shows the Euclidean heuristic cost of all states as color codes from green to red. Green being the lowest and Red being the highest with goal state shown as red arrow. Start state x_S is the first state to be explored hence added to the list with cost-to-come C(x_S)=C$_S$. C$_S$ is the cost for reaching the start state. It is calculated as the Euclidean distance from the start pose to start state.

Open list is sorted according to the goal cost f(x) = C(x) + H(x, x_G) which is an estimate of the cost to goal. Cost for movement from x to x′ is given by C(x,x′) which is explained through the algorithm.

The algorithm runs in a while loop until the goal is found or open list is empty. At the start of the loop, a state with the lowest cost is removed from the open list and added to the closed list. If this removed state x is the goal, then the loop ends and the path is traced back to the start as shown in steps 9 to 11 of algorithm 5.7.

Neighbours are explored if the goal is not reached. If the neighbour is an unvisited state i.e. not in open list or closed list, then the cost-to-come of the neighbour is updated as C(x′)= C(x)+ C(x,x′) + Obs(x′). Obs(x′) is the cost of the obstacle in x′ which will be discussed in section 5.3. For tracing the path back to start, current state x is marked as parent state of x′. This is shown in steps 15-18 of algorithm 5.7.

If the neighbour state is already in the open list i.e. alive state, then cost-to-come through x to x′ which is C(x)+ C(x,x′) should be less than the neighbours cost-

```
1: Get $x_S$ and $x_G$
2: Upated Heuristics cost H(x,$x_G$)
3: Upated cost-to-come C(x$_S$)
4: Add $x_S$ to OpenList
5: **while** OpenList not empty **do**
6:    x=GetLowCostState(OpenList)
7:    Remove x from OpenList
8:    Add x to ClosedList
9:    **if** x==$x_G$ **then**
10:       return Path
11:   **end if**
12:   **for** each neighbour x′ of x **do**
13:       **if** x′ in OpenList and C(x) + C(x,x′) + Obs(x′) < C(x’) **then**
14:           Mark x as parent and update cost C(x’) = C(x)+C(x,x′) + Obs(x’)
15:       **else if** x′ not in ClosedList **then**
16:           Mark x as parent and update cost C(x’) = C(x)+C(x,x′) + Obs(x’)
17:           Add x′ to OpenList
18:       **end if**
19:   **end for**
20: **end while**
```

Figure 5.7 *Algorithm for A-Star Planning*

to-come C(x′). This implies a more efficient plan, thus the neighbour cost C(x′)= C(x)+ C(x,x′) and parent state are updated.

C(x, x′) provides the cost from state x to x′. If only Euclidean distance is considered, then the cost of going forward and reverse for the forklifts will be same. Even though the forklifts are omnidirectional, turning and switching directions increases the travel time. To avoid switching direction or sharp turns, the actual movement of the forklifts must be considered for a realistic cost estimate.

Map points have only two-dimensional location without any orientation. But the forklift movement has to consider turning cost as well. For this each state is assigned with the temporary orientation during planning. When assigning the parent states in steps 14 and 16 of algorithm 5.7, orientation is assigned to the child state. Absolute angle(arctan) between the parent state and child state is assigned as the heading for the child state.

For two states $x_1 = (x1, y1, h1)$ and $x_2 = (x2, y2, h2)$ where h1 and h2 are assigned temporary headings, equation 5.5 shows the cost calculation between the two states.

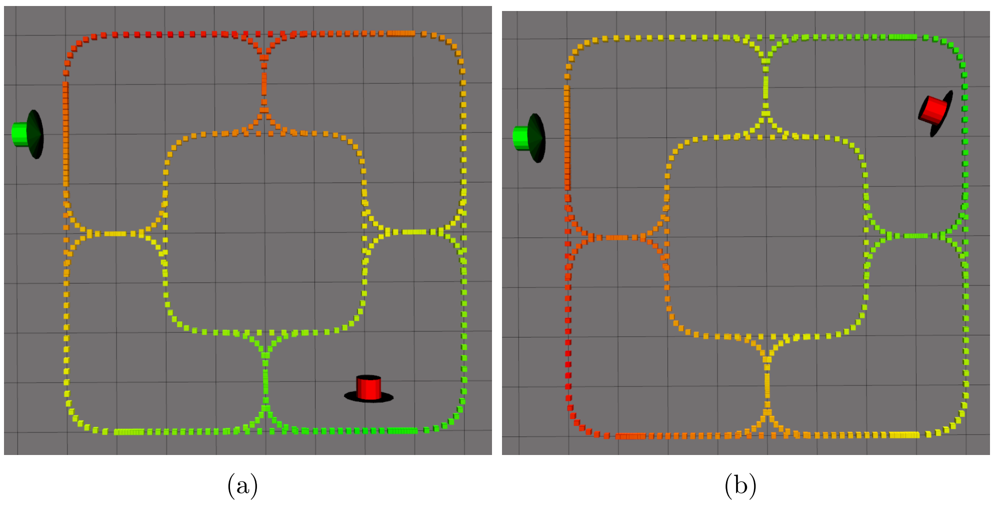


Figure 5.8 *Sample Euclidean heuristic cost map with different goals.(Goal is shown by the red arrow. In the color map, Green indicates lower cost and increases to red being the highest cost)*

Weights w_d and w_h are used to tune the priority for each cost. Note that $h2$ is not used as this assigned heading is from a different parent.

$$C(x_1, x_2) = w_d * Euc(x_1, x_2) + w_h * |norm(h1 - \arctan(y2 - y1, x2 - x1))| \quad (5.5)$$

where $Euc(x_1, x_2) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$ is the Euclidean distance and $norm(h)$ is the angle normalized between $-\pi$ and π.

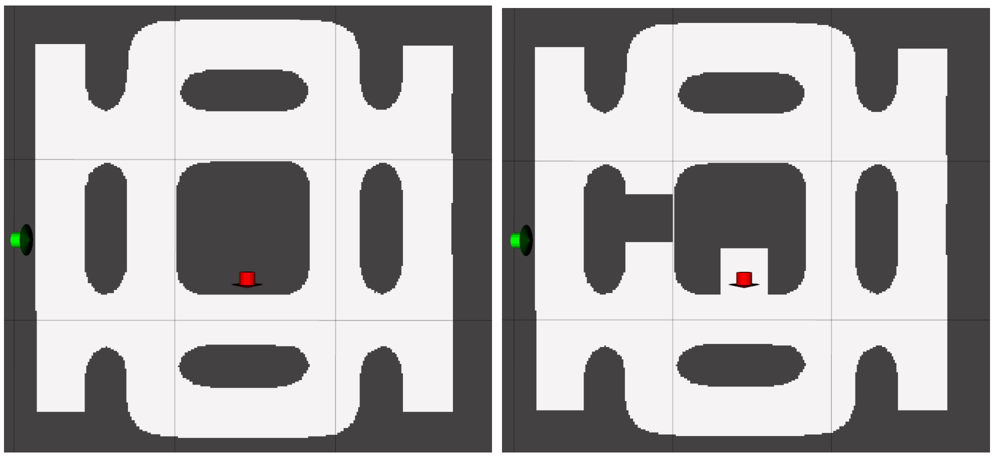
For the states there is no assigned heading (for example, start state), h1 is invalid, Hence heading cost is zero and $C(x1, x2) = w_d * Euc(x_1, x_2)$.

5.3 Obstacle Tracking and Implementation

In the previous section, obstacle cost obs(x) is not provided for the planning. For this obstacles and other forklifts in the area must be tracked. As mentioned before this information is provided by the network because of lack of onboard sensors. Future addition of range sensors is considered in the design of obstacle tracking. Range sensors are a type of sensors that can detect the presence of obstacles from a distance.

Provided obstacle information is divided into two categories:static and dynamic obstacles. Static obstacles are similar to any new changes to the map information or long-term disruption in the path e.g. construction work. This is simulated and provided by the user in order management system. Dynamic obstacles are analogous to moving objects that change their location over time. Location of other forklifts is received through the server to simulate this situation. This can be substituted by range sensors in future.

To enable easy addition of range sensors which scan the area for obstacles, an occupancy grid-based tracking is used. Occupancy grid store the information about obstacles in a discretized spatial space. Sensor uncertainty is observed in occupancy maps as probabilistic values. Since there are no sensors the details of the probabilistic occupancy map is left for future work. All the occupancy maps used contains binary occupancy values i.e. absence or presence of the obstacle is denoted by either 0 or 1 respectively.



(a) Occupancy Grid from Map Points (b) Occupancy Grid with Static Obstacles

Figure 5.9 *Comparison of map occupancy grid to static occupancy grid*

To avoid the robots to go outside the roads, the map information is first included in the occupancy grid. This is done by setting the values of map points with a square of side equal to road width. Since the map points are sufficiently close to each other, this creates a traversable occupancy map. Figure 5.9(a) shows map occupancy grid created from map points.

There are two types of occupancy grid used in the planning. Static occupancy grid maintains the current state of the map and static obstacles which is used by the planner. This map changes occasionally whenever any new static obstacles are added or the target loading areas are changed. Dynamic occupancy grid is the one which tracks all the moving and static obstacles. This grid is used for the local planning to avoid obstacles dynamically. Map information is added to both static and dynamic grid and changes when the map of the area is changed.

Information from the map is the fixed reference map. Static obstacles and target loading area changes overlayed on top of map grid to get static occupancy grid.

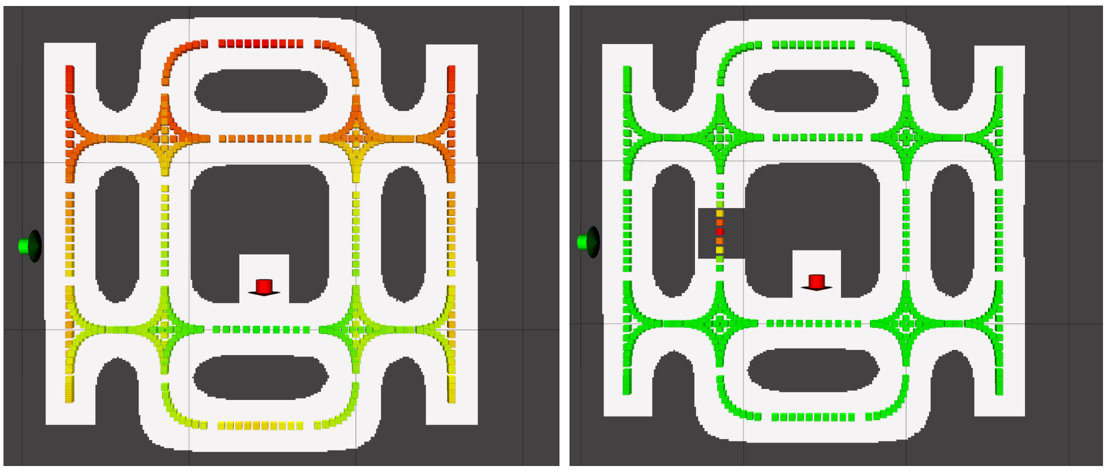
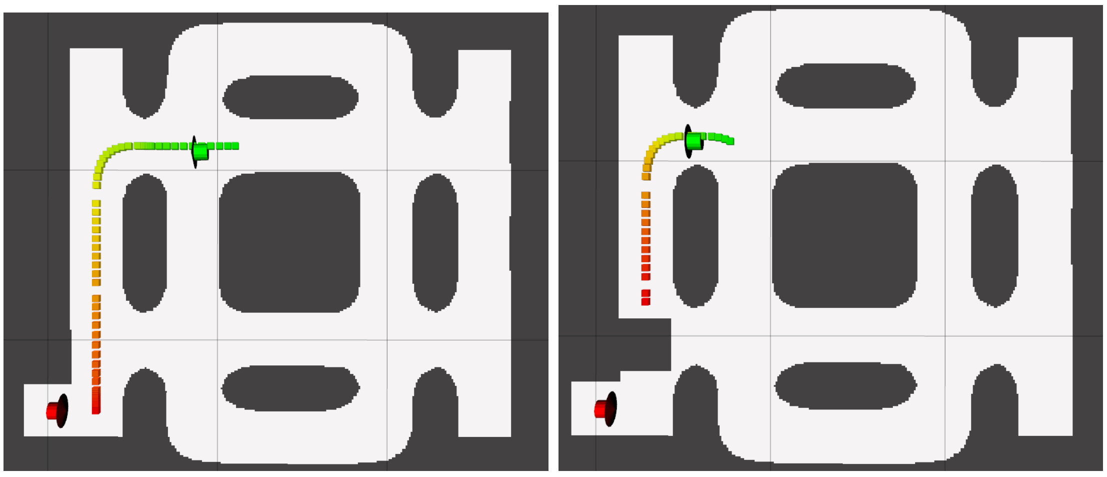

(a) (b)

Figure 5.10 *Cost map of the static grid*

Loading areas are outside the roads and to enable collision check in these areas, they have to be valid in the occupancy map. For this, the loading areas and static obstacles are added to the existing map information. This occupancy grid is used for calculating the obstacle cost.

Figure 5.9(b) shows a target loading area added as a free space and a static obstacle created as a non-free space. Obstacle cost obs(x) at a map point x is taken from this static occupancy grid. Assigning high cost(for eg. obs(x)=100) to map points in the obstacle area provides high traversing cost to obstacle areas. Figure 5.10 shows the estimated goal cost f(x) used for sorting the open list in the graph search. From this it is obvious that the map points that fall in the obstacle area are pushed to the bottom of the open list queue.

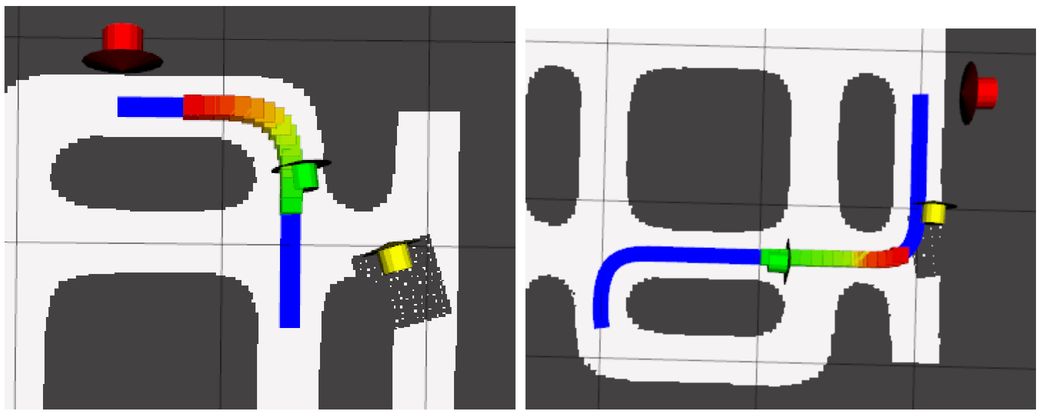
With high cost for obstacles, obstacle-free paths can be found if available. If no obstacle free paths are available, then the forklift cannot move from its place and can block other forklifts. Hence, map points that fall in the obstacles are not completely avoided but not preferred in global planning. If the global planner finds a plan



(a) Global planning without obstacles (b) Plan ends before obstacles when there is no alternate path

***Figure* 5.11** *Global Planning with and without obstacles*

through an obstacle if all path are blocked, the plan is created until the obstacle. This case creates paths through obstacles if all the paths are blocked. This feature is specifically enabled for the demonstration for allowing the forklifts to reach the obstacles and wait for it to change. Figure 5.11 shows the global planning in the cases where there is no static obstacle and planning until the obstacle if there is no alternative.

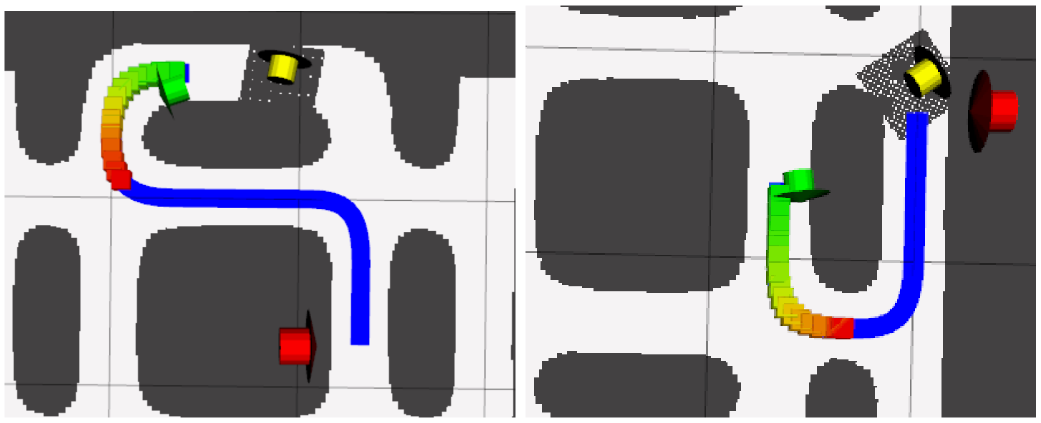


(a) Global and local plan without collision (b) Global plan with collision but local plan without collision

***Figure* 5.12** *Local Plan used from global plan for collision avoidance(Blue line denotes the global plan and Green to Red is the local plan)*

To avoid dynamic obstacles, the global plan is used to create a local plan and checked for collision in the dynamic grid. Local plan is the selected as the plan from the current location to short distance based on the forklift speed. For slow speeds a distance of 1m is used. This implies the collision is checked for the next 1m from the current location of the forklift. This plan is checked with dynamic grid for collisions.

Information about other forklifts in the area is tracked in the dynamic grid. Figure 5.12(a) shows a simulated dynamic obstacle and its tracking in dynamic grid.



(a) Global planning avoiding other forklift (b) Alternate path to goal

Figure 5.13 *Global replanning with dynamic Obstacles*

The local plan avoids nearby dynamic obstacles by adding a temporary static obstacle to the global planner, thus requesting for a new plan. This temporary obstacle is tracked and removed when the status changes, thus freeing the area if there is no obstacle. Figure 5.13 shows different situations where the replanning of global plan is requested.

6. TASK EXECUTION

This chapter explains the controller implementation of path following, pallet picking and placing algorithms. Path following executes the local plan from the path planner by applying control motions and guide the forklift along the plan. Path planner does not provide any paths for picking and placing the pallets. Separate controller is implemented for pallet picking independent of path following(and path planner).

Section 6.1 describes the path tracker algorithm implemented in the forklifts. The controller also has the ability to move the forklift to a given pose i.e.path with single point and heading. This enables to move the forklift to particular orientation and thus used for searching the pallet as the target location is provided by the order. Path following controller is implemented in differential drive mode without using the omnidirectional movement(mecanum driving)for smoothness.

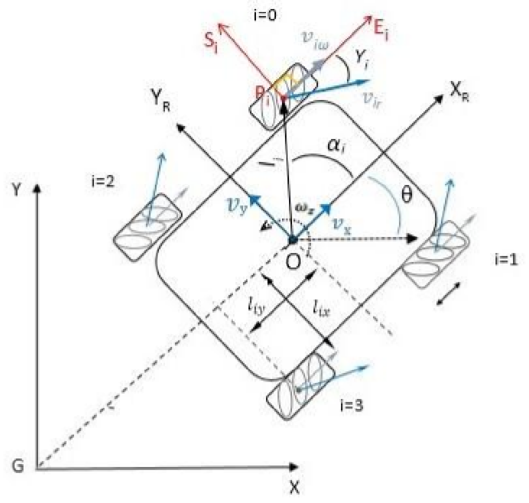


Figure 6.1 *Wheel Configurations for mecanum wheel*
[21, Fig.1]

Section 6.2 shows the implementation of pallet picking algorithm using marker input from the camera without using positioning with markers. After detecting the markers on the pallet, the algorithm is designed such that the marker always stays visible to the camera. Omnidirectional movement is utilized extensively for this function.

Path Tracker module implements all these different functions during different stages of demonstration. To manage the different scenarios and execute the tasks, a state machine is implemented in section 6.4.

6.1 Path follower

Given the path or target pose, the objective of this section is to design a smooth controller that can reach the follow and reach the target with minimum error.

Control values which is the rotational speed of all four wheels($\omega_1, \omega_2, \omega_3, \omega_4$) of the forklift can be calculated from the linear and angular velocities using kinematic model. For kinematic modelling of the forklift, few assumptions are made. The wheels are assumed to be fixed to the forklift body as shown in Figure 6.1 and can only rotate about their rotational axis. Wheel slip is assumed to be zero. If linear velocities of the forklift x and y direction in the body fixed frame is in v_x and v_y, then the wheel velocities are given by equation 6.1[21, eq. 20].

$$\begin{aligned}
\omega_1 &= \tfrac{1}{r}(v_x - v_y - (l_x + l_y)\omega) \\
\omega_2 &= \tfrac{1}{r}(v_x + v_y + (l_x + l_y)\omega) \\
\omega_3 &= \tfrac{1}{r}(v_x + v_y - (l_x + l_y)\omega) \\
\omega_4 &= \tfrac{1}{r}(v_x - v_y + (l_x + l_y)\omega)
\end{aligned} \tag{6.1}$$

where l_x and l_y are the distance of the wheels from centre of the forklift in x and y direction. Angular velocity of the forklift is given by ω and r denotes the radius of the wheel.

For the simplicity and smoothness of designing path controller, the path follower is implemented in differential drive mode i.e. $v_y = 0$ during path following. Then the objective of the controller is to determine v_x and ω. The speed limits of the linear and angular velocity limits of the forklifts can be calculated from the speed limits of the servo motors. Let v_{max} and ω_{max} be the linear and angular limits for the controller.

Geometric path controllers are the simplest form of control methods. These methods

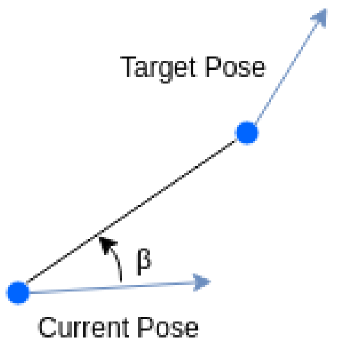


Figure 6.2 *Path Follower Error Variable β*

considers the geometric relationship between the path and the current location for control. Pure-pursuit controller is one of the most common method used in geometric tracking. Pure-pursuit controller calculates the desired curvature of the vehicle from the current pose to a specific target pose on the path. This target pose is chosen at a distance ahead of the vehicle. It pursues a moving point on the path, hence the name.[22]

$$v_x = v_{max}.(\omega_{max} - |\omega|) \tag{6.2}$$

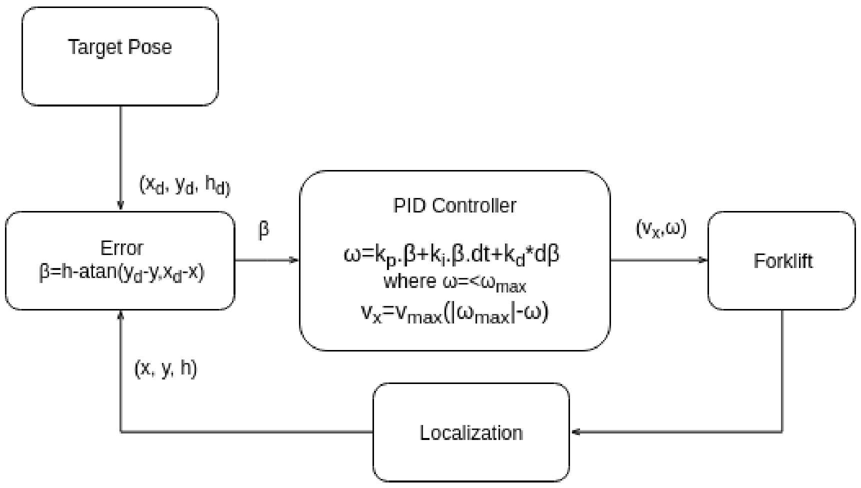


Figure 6.3 *Control system for path follower*

Path following is always assumed to be moving in forward direction i.e.$v_x >= 0$ and a smooth relationship between linear and angular velocity is created. Value of v_x is set to be dependent on the angular velocity and inversely proportional i.e.$v_x = 0$

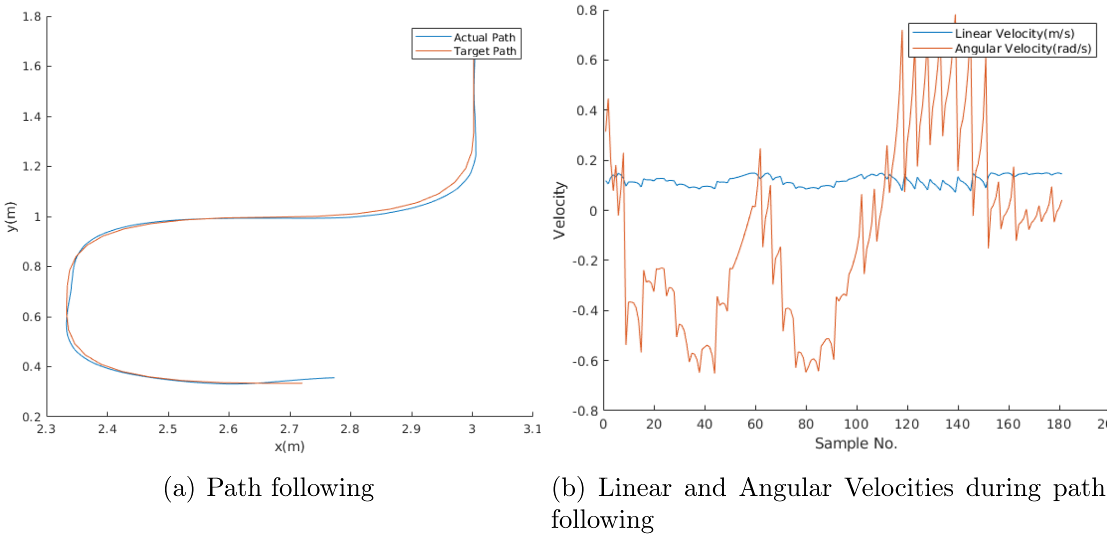
if $\omega = \omega_{max}$. In practice, this allows the forklifts to turn in place if the maximum angular velocity is set and slow down during turning.

Instead of calculating the curvature as in pure pursuit method, deviation of heading from the target(β) is directly used as the error. Geometrically this error turns the forklifts towards the target. Figure 6.3 shows the control system used for the path following. Turning error(β) is calculated from

$$\beta = h - atan(y_d - y, x_d - x) \tag{6.3}$$

where current vehicle pose is given by (x, y, h) and target pose is given by (x_d, y_d, h_d).

The desired target point is selected from the path using a minimum distance from the current position similar to pure-pursuit. With the desired target pose, the deviation angle error β is calculated from the current pose. This deviation error is tuned using a PID controller to get the angular velocity. Parameters for the PID controller are found by trial and error method. The tuned angular velocity is limited to ω_{max} and v_x is calculated from equation 6.3. Figure 6.5(a) shows the target and followed

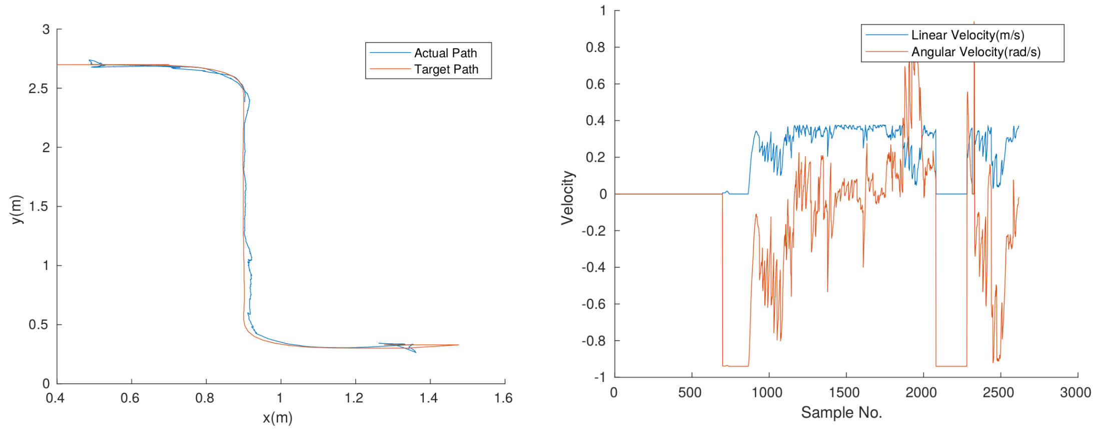


(a) Path following

(b) Linear and Angular Velocities during path following

Figure 6.4 *Path following performance in simulation(assuming perfect tracking and localization)*

path by the controller from the simulation. Note that the localization is assumed to be without error in the simulator. The tracked path is within the error tolerance for the forklift. Control velocities shown in figure 6.4(b) are not smooth when the target pose switches during tracking and having a continuous smooth path could



(a) Path following

(b) Linear and Angular Velocities during path following

Figure 6.5 *Path following real-time performance with the position solution from EKF*

avoid this problem.

Figure 6.5 shows the performance of the controller in real-time with position from EKF. It is not the actual position of the forklifts. Figure 6.5(b) shows the control velocities. The tracker is aggressive in its tracking and thus not smooth when there is jump in position solution and could be improved further.

The controller does not stop at the end of the path and thus stopped when the heading error(heading difference between desired and actual pose) is close to zero. During the pallet search the same controller is used. Searching is executed by providing a local target in front of the pallet to the controller. The controller aligns the forklift such that the pallet is within the view of the forklift camera.

6.2 Pallet Picking

When the forklift is aligned to the view the pallet, pallet location is provided by marker detection using image processing. Markers are black and white bitmap images that can be uniquely identified from an image. The implementation uses one of the most used aruco marker detection from opencv library. Marker detection can provide the defined unique id of the marker and its relative pose from the camera.

Figure 6.6 shows the parameters provided by the marker detection. Distance of the marker from the camera(d), deviation angle of marker from the centre(α) and orientation difference between marker and camera(γ) are the parameters determined from each detected marker. Control for this part is implemented using visual servoing i.e. driving only using camera inputs. For this the algorithm must maintain the marker within the viewing angle of the camera at all time.

Figure 6.7 shows the algorithm used for pallet picking. α_{min},γ_{min} and d_{tol} are the deviation tolerance for maintaining the marker in the centre of the camera view. For maintaining the marker within the viewing angle, α which is the viewing angle to the marker must be minimized during the movement. Control velocities v_x, v_y and ω which are explained in 6.1 are to be calculated from the marker location parameters.

Equations 6.4 shows the error values and their mapping to the control values. Algorithm aligns the forklift to the pallet by minimizing α and γ to zero and d to d_{min} in the same order where d_{min} is the minimum distance required for the marker detection to work. d_{min} also prevents the forks from hitting the pallet during the maneuver. For the demonstration d_{min} used is 15cm. Once aligned the pallet can be picked by driving straight through without any marker input. This solves the problem of missing markers when the pallet is close to forklift.

$$\begin{aligned} v_y &= v_{miny}.(\frac{\alpha}{|\alpha|}) \\ \omega &= \omega_{min}.(\frac{\gamma}{|\gamma_{min}|}) \\ v_x &= v_{minx}.(\frac{d-d_{min}}{|d-d_{min}|}) \end{aligned} \tag{6.4}$$

Forklifts are run at minimum constant speed during visual servoing to avoid blurring during marker detection. Let v_{minx},v_{miny} and ω_{min} be the minimum velocities of v_x,v_x and ω respectively. All the parameters are decoupled from each other and made

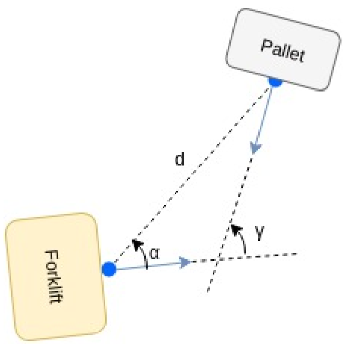


Figure 6.6 *Pallet location from marker detection*

```
1: while Pallet Not Picked do
2:     if $|\alpha|>\alpha_{min}$ then
3:         $v_y = v_{miny}.(\frac{\alpha}{|\alpha|})$
4:     else if $|\gamma|>\gamma_{min}$ then
5:         $\omega=\omega_{min}.(\frac{\gamma}{|\gamma_{min}|})$
6:     else if $|d-d_{min}|>d_{tol}$ then
7:         $v_x=v_{minx}.(\frac{d-d_{min}}{|d-d_{min}|})$
8:     else
9:         Drive towards pallet
10:    end if
11: end while
```

Figure 6.7 *Algorithm for Pallet picking*

proportional to the control values. Geometrically applying $v_y = v_{miny}$ reduces α by moving the forklift left and right. γ can be reduced by rotating the forklift(applying $\omega=\omega_{min}$) to align with the pallet. When the forklift moves too close to the pallet i.e. $d < d_{min}$ $v_x = v_{minx}$ reduces the error.

6.3 Pallet Placing

After the forklift reaches the end of the path during pallet drop-off, the target drop-off location must be reached without going outside the restricted area. For this no planning is implemented. Instead the omnidirectional feature of the forklift is utilized. To minimize the forklift movement outside the roads the forklift is aligned towards the target so that the forklift can drive straight towards the target.

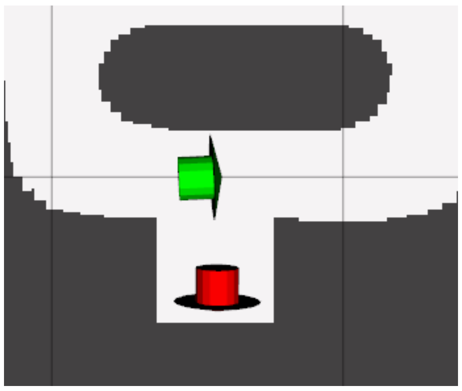


Figure 6.8 *Target for pallet placement with restricted movement(Current pose is indicated in Green and Target pose in red)*

Sensor fusion localization solution is used for the controller and the target pose is provided by the order as explained before. If (x,y,h) defines the current pose of the vehicle and (x_d, y_d, h_d) denotes the target pose, then the errors in each state variable is given by $d_x = x - x_d$,$d_y = y - y_d$ and $d_h = h - h_d$. If the heading error dh is zero, then the forklift is in the same orientation as the target. Then the problem becomes simple for the controller. The forklift can move left and right using x-error dx to be in straight line with the target and finally moving towards the target with dy without going out of the restricted area. So the errors dh,dx and dy are brought to zero one by one in the same order instead of simultaneous movement. If implemented in this order, then the velocities become directly proportional to the errors as given by equations 6.5.

$$\begin{aligned} \omega &= -ad_h \\ v_y &= -bd_x \\ v_x &= -cd_y \end{aligned} \tag{6.5}$$

where a,b and c are gains for each of the errors.

6.4 State Machine

The path tracker module implements all the algorithms specified before and they must be integrated to work together during different stages. To execute the tasks a state machine is implemented for switching between tasks. State machine is model that has a finite number of states and switches between different states based on conditions. The state machine can be in one state at a time. This finite state machine is used to switch between different controllers. Figure 6.9 shows the state machine of the path tracker module.

At the start, the tracker waits for the position from sensor fusion and initializes to idle state(STATUS_IDLE). When an order is sent from the server, it is accepted by the tracker if it is in idle state. Receiving the order switches the state to path tracking state(STATUS_TRACK_PATH) for movement to the pallet picking area. Path planner is requested for a plan to the target picking area. In this state, whenever a new path is sent by the planner it is followed using the path follower without exception.

When the path ends which is indicated by the planner, the tracker starts to search for the pallet in pallet search state(STATUS_SEARCH_PALLET). If the pallet is

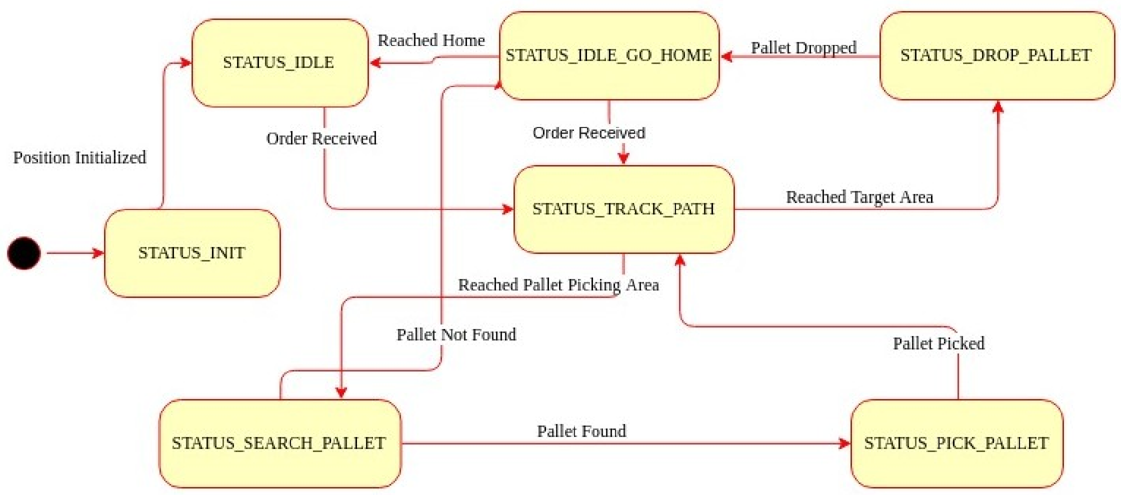


Figure 6.9 *State Machine for task execution*

not found after the search maneuvers, the area is reported empty and switches to idle state (STATUS_IDLE_GO_HOME). The forklift moves to the start location by requesting a plan to the target. The behaviour of this state is same as the tracking state except new orders are accepted at this state.

If a pallet is found, then it is switched to pallet picking mode(STATUS_PICK_PALLET) where control using camera-based markers is used for picking the pallet. After successful completion of pallet picking, new path for the drop-off location is requested from the planner in tracking mode. When the path ends, the drop-off maneuver is executed in drop pallet mode(STATUS_DROP_PALLET). After all the tasks are executed, the forklift reaches home position by requesting a path to the same and waits for the next order.

7. SYSTEM INTEGRATION AND RESULTS

The algorithms are tested separately in individual modules and are combined to execute tasks in real-time. These modules can communicate with each other and some modules are dependent on each other for their own behaviour. Section 7.1 describes the communication messages and interaction between the different modules in the system for real-time execution. Demonstration results are described in section 7.2. The overall objective of the demonstration is reviewed and the actual results were evaluated for their performance.

7.1 System Integration

Given the pallet movement order, the tasks must be executed by the forklifts. Execution of demonstration tasks require complex communication between different modules as specified before. This complex communication is best represented in a Unified Modeling Language(UML) diagram. Figure 7.1 shows the UML diagram of communication between the modules.

To execute the tasks wheel velocity commands must be sent to Servo Drive module. The frequency of the command determines the responsiveness of the forklift to the changes in the environment. Position solution from the sensor fusion is of high frequency(70Hz) which is same as the IMU frequency. This frequency is sufficient for the control. Hence all the control actions are triggered by the position solution except when picking the pallet. For this function, a camera based trigger control is implemented for pallet picking which is of 30Hz frequency. The camera module is active during the pallet picking state and stays idle otherwise.

State machine and Path tracker are implemented in the same module to enable a single point control for the servos. This module provides all the control actions to the servo drive. State machine manages all the tasks for execution and requests for information from other modules. State machine requests path to picking location

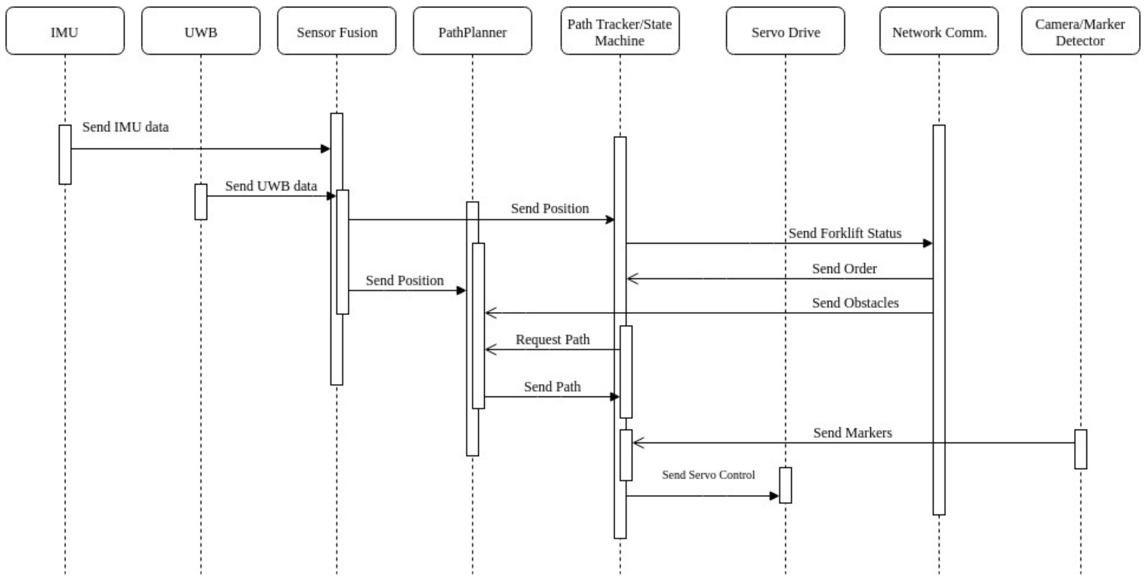


Figure 7.1 *UML diagram of communication between different modules*

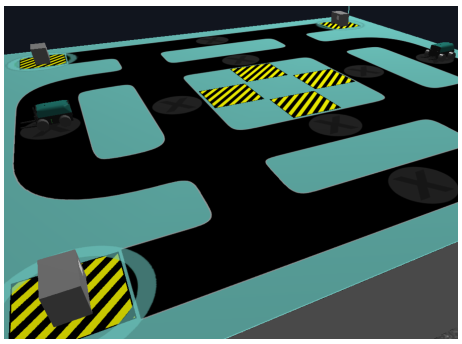
from path planner when a new order is received or path to drop off location when the pallet is picked.

Path planner receives the position information from sensor fusion and obstacle information from the network. When the planner sends a path to the tracker, it continuously checks for collision and provides an alternate path if available. Path planner stays idle if there is no request or no active target is being executed.

7.2 Demonstration Results

The demonstration is initialized with the forklifts aligned to zero heading. They can be started from any location but the heading is fixed. In the future having a heading sensor can solve this issue and the forklifts can start from any pose. Figure 7.2(a) shows the forklifts waiting for the orders. Orders are assigned to any free forklift without any particular order. Movement order is provided through this interface by the user. Figure 7.2(b) shows the one of the forklifts has accepted the order and moving towards the pallet for picking.

The forklifts were able to stay inside the roads most of the time. They sometimes ran close to or over the edge of the roads because of tracking errors or localization errors if the line of sight of the UWB beacons is blocked.


(a) Forklifts in starting position waiting for order

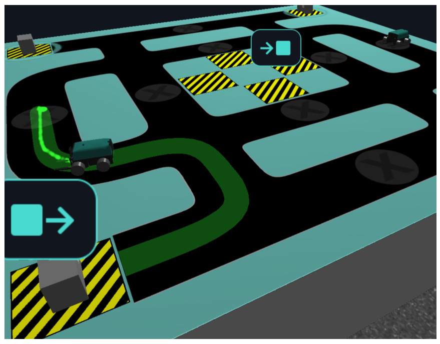
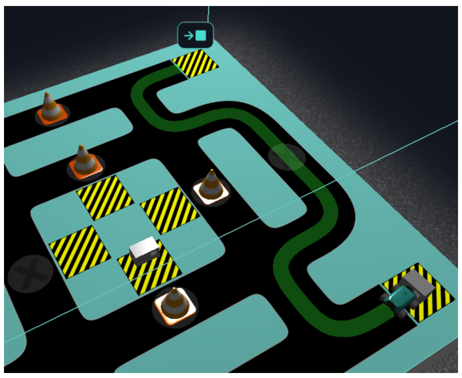
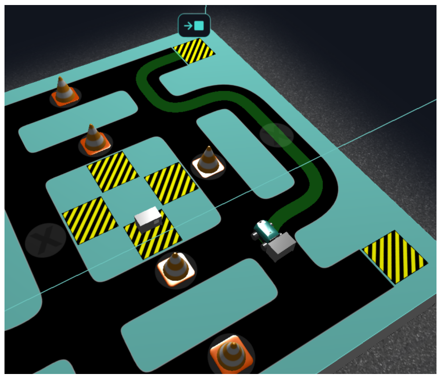
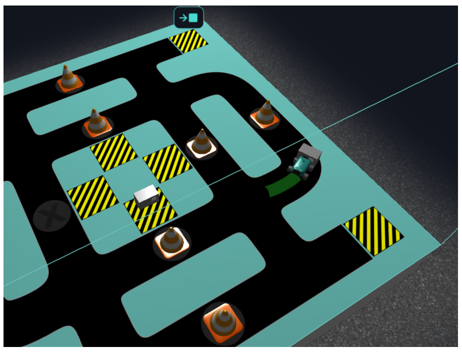

(b) Order assigned to one forklift and the other being idle

Figure 7.2 *Starting Position of forklifts during the demo*


(a) Planning to target through free path


(b) Forklift on route to target


(c) Stopping before obstacle and wait for it to change

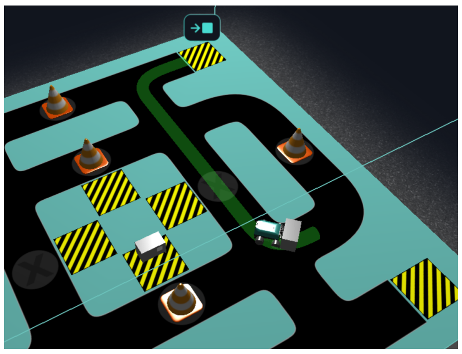
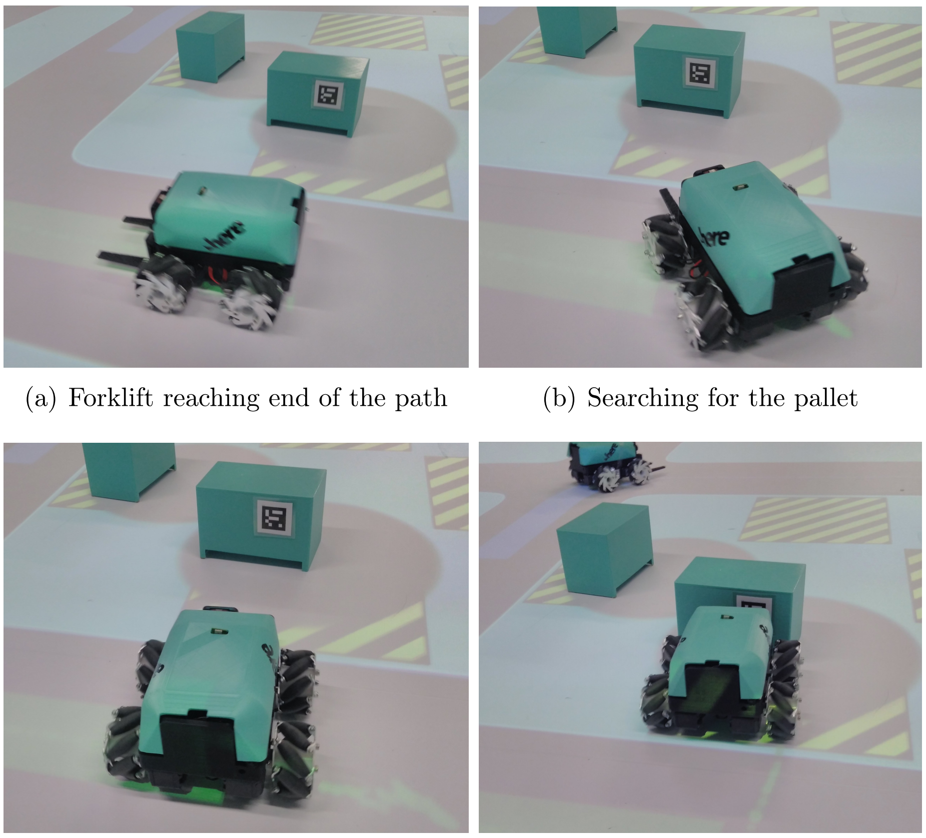

(d) Planning through a different free route

Figure 7.3 *Planning with static obstacles*

User inputs of static obstacles are integrated in to the path planning and responsive for changes in the user inputs. Figure 7.3 shows planning with different user inputs of static obstacle. Planner creates path always avoiding the static obstacles. If the no path is available as shown in figure 7.3(c), plan is created towards the closest obstacle and wait for any path to be free.

Multiple robots were able to run without colliding if the network communication is robust. The collision avoidance fails if there is a delay in receiving other forklifts position. Since the collision avoidance is based on the global plan, they are not without errors. There are cases where the forklift runs without any plan such as pallet picking and placing. These combined errors could cause collision during the execution.

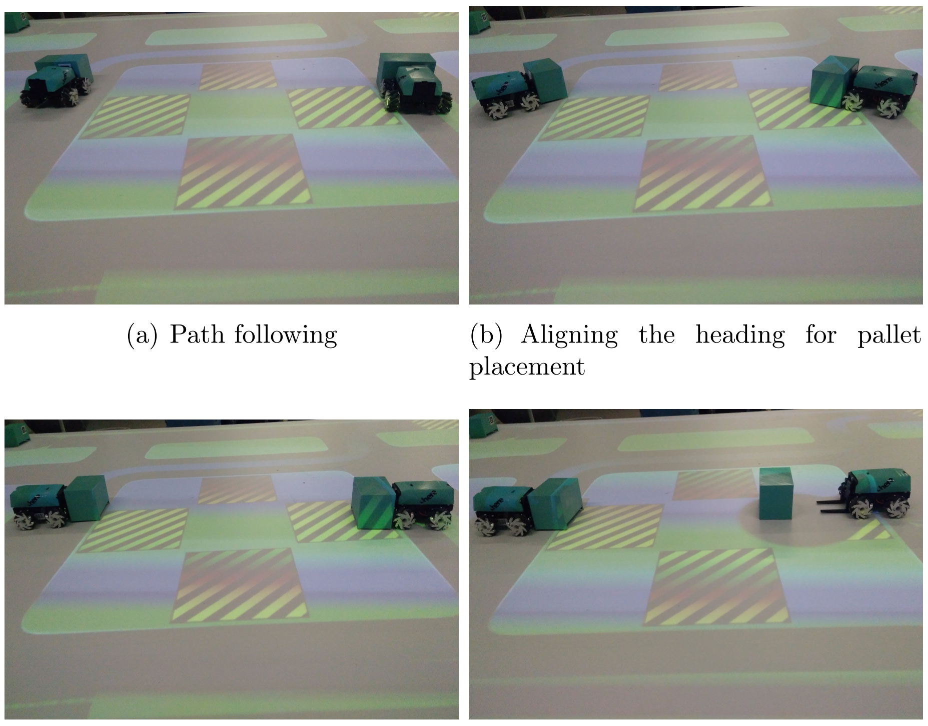
Addition of onboard obstacle sensors provides better visibility of the forklift surroundings and hence these issues can be avoided. Dynamic planning could be independent of the plan and can be based on the predicted trajectory of the forklift. The platform and algorithm could easily add these changes without many changes to the system.



(a) Forklift reaching end of the path (b) Searching for the pallet

(c) Aligning with the detected pallet (d) Picking the pallet

Figure 7.4 *Different stages of autonomous pallet picking*



(a) Path following

(b) Aligning the heading for pallet placement

(c) Aligning the offset

(d) Placing the pallet

Figure 7.5 *Two forklifts placing the pallets on their target independent of each other*

Performance of the pallet picking and placing algorithms were within the demonstration requirements. Forklifts were able to detect and pick the pallets autonomously. Figure 7.4 shows the sequence of maneuvers executed by the forklift for picking the pallet autonomously. After reaching the end of the path the forklifts searches for the pallet and reports the space empty if no pallet is found. Better lighting is needed for the aruco marker detection to work. Pallet picking could fail if the pallets are not properly lit.

Pallet placement depends on the localization and accurate if the localization error is less. Localization errors could cause errors in the pallet placement but the robust pallet picking algorithm was able to compensate for the errors and align itself towards the pallet for picking.

Overall, the demonstration was able to evaluate and demonstrate the localization capability of the UWB positioning. The software system was robust throughout the demonstration and can be easily scaled to different autonomous systems. Its scalability and modularity features serve as an excellent platform for development.

8. CONCLUSION

The objective of the demonstration to implement a small-scale factory environment using UWB positioning is achieved. The forklifts were built in-house using 3D printing techniques and off the shelf hobby robot parts. Robots were able to locate themselves in the area using UWB positioning. UWB position and IMU are integrated in the sensor fusion using extended Kalman filter to provide robust localization.

Using wheel encoders which provide feedback about wheel rotation can improve the dead reckoning performance and better localization performance. The absolute initial heading is not determined by the current sensor systems. Additional sensors like magnetometer can provide absolute heading of the forklifts.

Robots were able to pick and move pallets between loading areas using orders from a centralized server. Execution of tasks is managed by a state machine. State machine runs in different modes and uses different controller for each mode. State machine sends the path request to the path planner and provided path is executed by path following controller in differential drive mode. Omnidirectional feature of forklift is used for implementation of robust pallet picking and placing algorithms. State machine reports the status of the execution continuously to the network and also the availability of pallets. Thus the server has the updated map always.

The localized forklifts use A* graph planning and obstacle information to create plans for routes using graph maps in OpenDRIVE format. Local planning for collision avoidance is implemented with global planning and replanning algorithm tries to create a balance between the number of replannings and avoiding obstacles. Current systems maintain large safe distance because of lack of onboard obstacle sensors. Static obstacles and other robot location is provided by the network and failure in communication could break the collision avoidance if the network fails. Having onboard obstacle sensors could enable to create more robust and dynamic local planning. The objective to demonstrate the UWB positioning in small-scale is achieved successfully.

BIBLIOGRAPHY

[1] A. M. Hossain and W.-S. Soh, "A survey of calibration-free indoor positioning systems," *Computer Communications*, vol. 66, pp. 1 – 13, 2015.

[2] S. Nirjon, J. Liu, G. Dejean, B. Priyantha, Y. Jin, and T. Hart, "COIN-GPS : Indoor Localization from Direct GPS Receiving," *MobiSys '14*, pp. 301–314, 2014.

[3] D. Dardari, P. Closas, and P. M. Djuric, "Indoor tracking: Theory, methods, and technologies," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 4, 2015.

[4] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen, "A Realistic Evaluation and Comparison of Indoor Location Technologies: Experiences and Lessons Learned," *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pp. 178–189, 2015.

[5] P. Davidson and R. Pich, "A Survey of Selected Indoor Positioning Methods for Smartphones A Survey of Selected Indoor Positioning Methods for Smartphones," *Ieee Communications Surveys & Tutorials*, vol. 19, no. c, pp. 1347–1370, 2016.

[6] D. H. Stojanović and N. M. Stojanović, "Indoor Localization and Tracking: Methods, Technologies and Research Challenges," *Facta Universitatis, Series: Automatic Control and Robotics*, vol. 13, no. Iii 43007, pp. 57–72, 2014.

[7] Z. Farid, R. Nordin, and M. Ismail, "Recent advances in wireless indoor localization techniques and system," *Journal of Computer Networks and Communications*, vol. 2013, 2013.

[8] A. Alarifi, A. Al-Salman, M. Alsaleh, A. Alnafessah, S. Al-Hadhrami, M. Al-Ammar, and H. Al-Khalifa, "Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances," *Sensors*, vol. 16, no. 5, p. 707, 2016.

[9] F. C. Commission, "Revision of Part 15 of the Commission's Rules Regarding Ultra-Wideband Transmission Systems," *First Report and Order in ET . . .*, pp. 1–118, 2002.

[10] S. Gezici, Z. Tian, G. B. Giannakis, H. Kobayashi, A. F. Molisch, H. V. Poor, and Z. Sahinoglu, "Localization via ultra-wideband radios: A look at positioning aspects of future sensor networks," *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 70–84, 2005.

[11] M. Kwak and J. Chong, "A new double two-way ranging algorithm for ranging system," *2nd IEEE International Conference on Network Infrastructure and Digital Content*, pp. 470–473, 2010.

[12] Y. Jiang and V. C. Leung, "An asymmetric double sided two-way ranging for crystal offset," *Conference Proceedings of the International Symposium on Signals, Systems and Electronics*, pp. 525–528, 2007.

[13] J. Xu, M. Ma, and C. L. Law, "Position estimation using UWB TDOA measurements," *IEEE International Conference on Ultra-Wideband*, pp. 605–610, 2006.

[14] J. E. M. Salih, M. Rizon, S. Yaacob, A. H. Adom, and M. R. Mamat, "Designing omni-directional mobile robot with mecanum wheel," *American Journal of Applied Sciences*, vol. 3, no. 5, pp. 1831–1835, 2006.

[15] S. Thrun, W. Burgard, and D. Fox, "Probabilistic Robotics," *The MIT Press*, 2005.

[16] J. Z. Sasiadek and P. Hartana, "Sensor data fusion using kalman filter," *Proceedings of the Third International Conference on Information Fusion*, vol. 2, pp. WED5/19–WED5/25 vol.2, July 2000.

[17] W. Koch, "Tracking and Sensor Data Fusion," *Springer*, 2014.

[18] S. M. LaValle, "Planning algorithms," *Cambridge University Press*, 2006.

[19] VIRES Simulationstechnologie GmbH, *OpenDRIVE References*, 2017. Available: `http://www.opendrive.org/references.html`.

[20] M. D. e.a., *OpenDRIVE Format Specification, Rev.1.4.* VIRES Simulationstechnologie GmbH, 2017. Available: `http://www.opendrive.org/docs/OpenDRIVEFormatSpecRev1.4H.pdf`.

[21] H. Taheri, B. Qiao, and N. Ghaeminezhad, "Kinematic Model of a Four Mecanum Wheeled Mobile Robot," *International Journal of Computer Applications*, vol. 113, no. 3, pp. 6–9, 2015.

[22] Y. Shan, W. Yang, C. Chen, J. Zhou, L. Zheng, and B. Li, “Cf-pursuit: A pursuit method with a clothoid fitting and a fuzzy controller for autonomous vehicles,” *International Journal of Advanced Robotic Systems*, vol. 12, no. 9, p. 134, 2015.