Chapter 4

Path Planning of Mobile Robot for Navigation using Vision Sensor

4.1 Introduction

Secure and efficient mobile robotic navigation requires an effective path planning methods because the robotic application is greatly affected by the quality of the path produced. There is plethora of seminal work available to get optimal path for various scenarios. Among all the available algorithms A* algorithm is very fundamental in nature and used for various static obstacle cases with the global map available. In this thesis path planning for mobile manipulator with static objects has been attempted.

Mobile manipulation systems are generally defined as mobile robots which have a robotic manipulator placed on them. A mobile manipulation system helps to overcome the limitations which both standard robotic manipulators and mobile robots face. A standard robotic manipulator has a fixed based and a limited workspace. The inverse kinematic solver hence fails to provide a solution for objects which are outside the manipulator workspace. Mobile manipulators allow the system to take up a wide array of poses and expand the reachable workspace. A standard mobile robot's path planner may fail to provide a feasible path in cases where obstacles are blocking the path of the mobile robot. In such cases, it may have to take up a costlier path to reach the goal location or in worst situations it may be stuck in local minima. A mobile manipulation system, on the other hand, allows the robot to grasp objects which are in the manipulator's feasible grasp range and can help in clearing the path.

The chapter is presented in following ways. Section 4.2 discusses the system description and section 4.3 provides the steps involved in vision-based navigation. Section 4.4 proposes an approach for a mobile manipulation based path planner which utilises a vision-based A* algorithm, for object clean-up operation. Section 4.5

analyses the results and discuss on the efficacy of the mobile manipulation system using suitable quantitative metrics. Section 4.6 summarises the research outcome from the simulations as well as real-world experiments which have been used to conclude the efficiency of the mobile manipulation based path planner.

4.2 System Description & Problem Statement

In this work, a mobile manipulation-based path planner is proposed which utilized a vision-based navigation system for the path planning. A multi-object based A* algorithm has been utilized for the process of object clean-up. A weighted cost function approach, which models the metric and spatial information of the objects, has been utilized for generating the priority of object pick-up. The overall system architecture which has been developed for the process of vision-based navigation is described in brief in order to understand the problem description. Each of the components will be further analyzed in detail. The key features of the systems have been explained as follows.

System description

1) An overhead monocular camera is utilized for capturing images of the environment. The images have a fixed resolution of 640×480 pixels.
2) The environment is well-lighted as scanty light will affect the performance of the vision system.
3) The obstacles in the environment are assumed to be static.

The problem statement can be summarized as follows.

Problem statement

1) Given a start and goal location, the mobile robot needs to avoid all the obstacles in its path and reach the goal location.
2) The path followed by the robot should be the optimal or shortest path for reaching the goal location while taking into consideration the physical constraints of the robot.
3) The robot will use vision-based features in order to localize itself. Figure 4.1 summarizes the entire flowchart for vision-based navigation which will be discussed in the detail in the further sub-sections of this chapter.

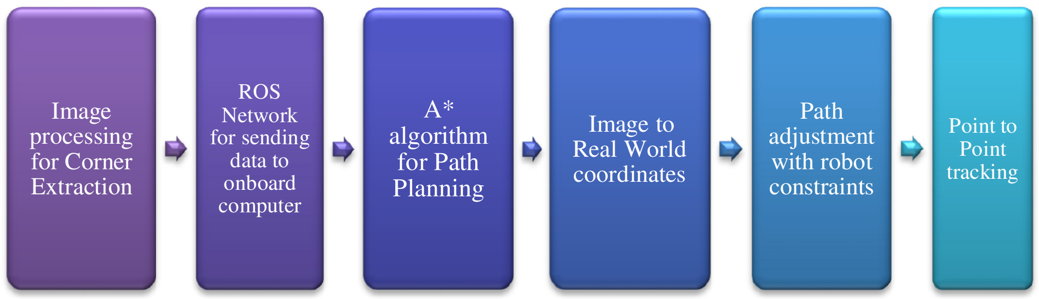
Image processing for Corner Extraction
ROS Network for sending data to onboard computer
A* algorithm for Path Planning
Image to Real World coordinates
Path adjustment with robot constraints
Point to Point tracking

Figure 4.1: Entire flowchart for vision-based navigation

4.3 Vision-Based Navigation

Vision-based navigation systems can be defined as systems which utilize computer vision algorithms to extract visual features from the world around the robot. These features of images are processed for robot localization and detect various obstacles obstructing the robot's path. In this case, the obstacles in the environment are assumed to be static obstacles.

4.3.1 Object Detection

The first step for the vision-based navigation involves the process of object detection. The key step in this process is differentiating the obstacles in the robot's environment from the free space which can be utilized for the process of path planning. The obstacles are separated from the background and undergo certain pre-processing steps in order to remove noisy pixels. These objects are then clustered and contour-based features are utilized to describe the location of the obstacle in (u, v) image coordinate space. The steps involved are explained in detail in the following sub-sections. The flowchart shown in Figure 4.2, summarizes various image processing techniques in object detection and corner extraction.

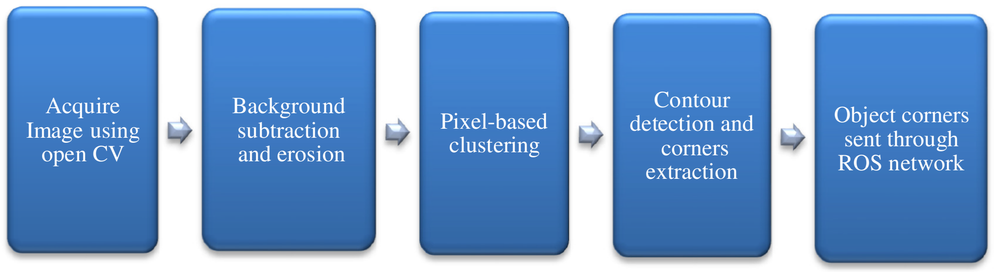
Acquire Image using open CV
Background subtraction and erosion
Pixel-based clustering
Contour detection and corners extraction
Object corners sent through ROS network

Figure 4.2: Flowchart of image processing to detect the center line

4.3.1.1 Image acquisition

The images for the experiment are acquired using a monocular USB camera which has a maximum resolution of 640×480. The camera is mounted on the ceiling to cover a wider field of view. The objects are placed within the camera's field of view. The camera is connected to a local computer which is running with Ubuntu 16.04. The camera is assumed to capture images in a well-lighted environment. The overhead camera utilized for object detection is shown in Figure 4.3.

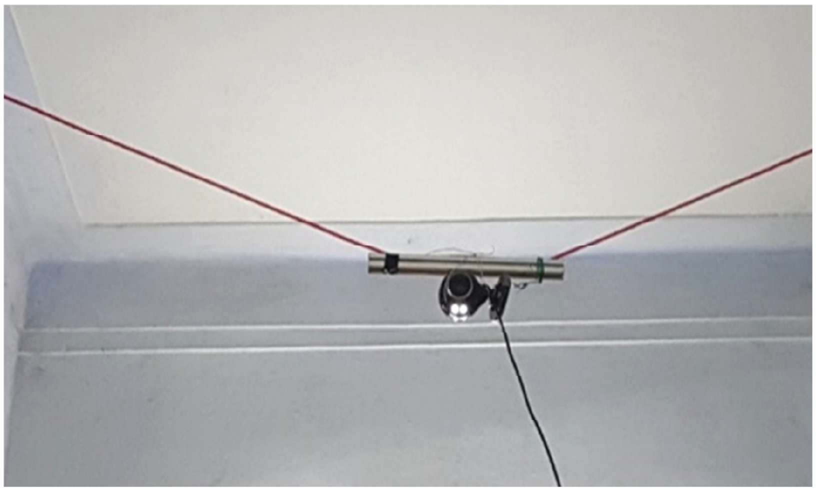

Figure 4.3: Overhead global camera for object detection

The overhead camera shown in Figure 4.3 has been calibrated in different orientations using 22 check board pattern and the MATLAB calibrator application. The results of the camera calibration were the stacked intrinsic and extrinsic parameters for the camera model. The intrinsic parameters include focal length (pixels), principal point (pixels), skew, radial distortion and tangential distortion. The extrinsic parameters include rotation vectors (3×3×22) and translation vectors (22×3) in cm. The mean projection error resultant from this process is 0.6049. Figure C3 (in Appendix-C) shows a sample image which has been utilized for camera calibration. Figure C4 shows the re-projection errors for the 22 images which have been used for the process of calibration. Figure C5 shows the views of the checkerboard configurations in Euclidean space as seen by the camera. Hence, using the process of camera calibration, the (u, v) coordinate values of optimal path are converted into real-world coordinates (x, y). These real-world coordinates will be used for giving actuation commands to the robot in real-world dimensions.

4.3.1.2 Image conversion

The images captured by the camera are converted from RGB format to grayscale, this conversion makes images easier to handle and better to work with for the process of thresholding and further pre-processing operations. The conversion from RGB to grayscale uses the following equation.

$$I_{\text{gray}}(x,y) = [0.299 \quad 0.587 \quad 0.114]\begin{bmatrix} I_{rgb}\{R\}(x,y) \\ I_{rgb}\{G\}(x,y) \\ I_{rgb}\{B\}(x,y) \end{bmatrix} \tag{4.1}$$

where, $I_{\text{gray}}(x,y)$ signifies the pixel values in grayscale format. $I_{\text{gray}}\{R\}(x,y)$, $I_{\text{gray}}\{G\}(x,y)$, $I_{\text{gray}}\{B\}(x,y)$ are the pixel values for the red, green and blue channels of the RGB image respectively. The input image of the map obtained from the camera in RGB format is shown Figure 4.4(a). The resulting image is obtained after conversion from RGB to grayscale is shown in Figure 4.4(b).

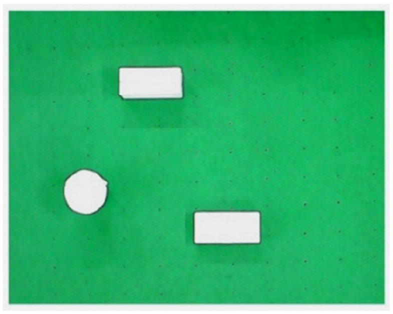
Figure 4.4(a): Original image in RGB format

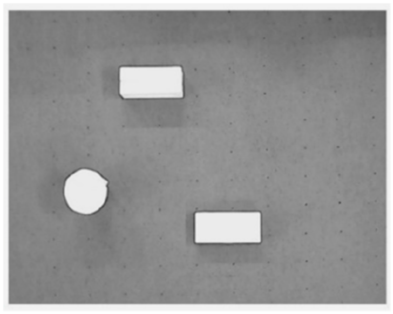
Figure 4.4(b): Image after conversion to Gray scale format

4.3.1.3 Background subtraction and erosion

The process of background subtraction is useful when obstacles are placed in an environment with a constant background and a varying foreground over a wide range of pixel values. A pre-determined binary threshold is applied to the image to separate the required obstacles in the image from the background. The resulting image after applying a binary threshold to the image is shown in Figure 4.5(a). Certain unnecessary pixels were generated due to uncertain lighting or sensor noise have been removed using an erosion operation. Erosion is a morphological operation which utilizes a structural element to remove isolated pixels generated due to noise. The image after an erosion operation is shown in Figure 4.5(b).

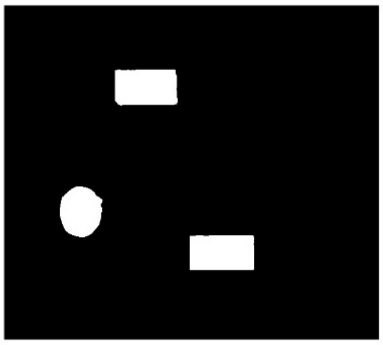
Figure 4.5(a): Image after thresholding

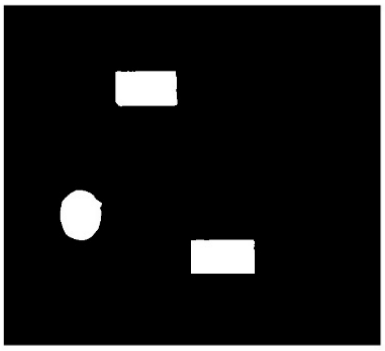
Figure 4.5(b): Image after erosion

4.3.1.4 Pixel based clustering

In order to separate one object from another and to count the number of obstacles, a clustering method is utilized. This clustering technique utilizes a disjoint-set data structure [Cormen (2009)] in order to separate one obstacle from another on basis of the Euclidean distance between pixels. Each cluster is given a separate colour than the other to differentiate them. The image obtained after the clustering operation is shown in Figure 4.6.

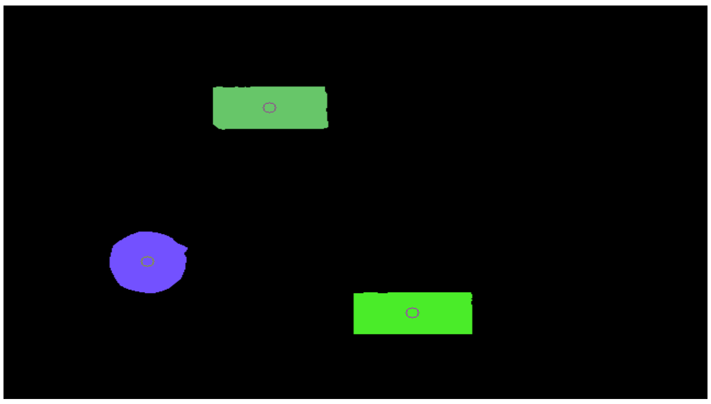
Figure 4.6: Image after clustering operation

4.3.1.5 Contour detection and corner extraction

Contour detection procedure is used to connect points of similar intensity in an image. Constructs like the Freeman Chains are utilized which have the current pixel information in the form of a step in a particular direction, which help to estimate the location of the next point on the same curve [Kaehler & Bradksi (2016)]. The corners of the detected rectangle contours are utilized to specify the location of an object in image coordinates space. The detected rectangle contours in the image are shown in Figure 4.7.

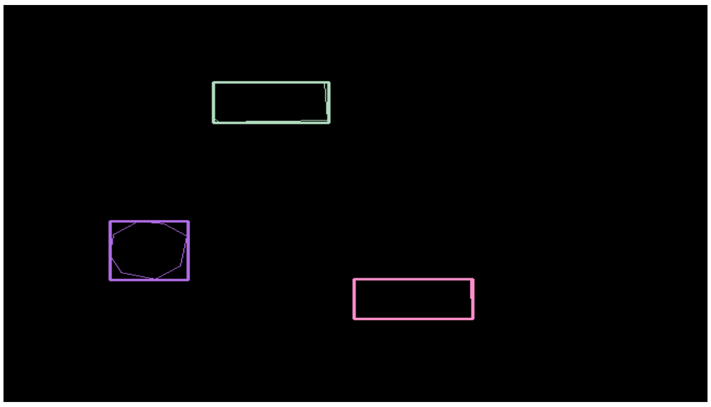

Figure 4.7: Image after contour detection

4.3.2 Network for Wireless Data Transfer

The image data is acquired from the overhead camera by an external computer running Ubuntu 16.04 and ROS Kinetic Kame using OpenCV 3.2. After the image processing steps described in the earlier section, the corners of the rectangular contours are received. The robot, on the other hand, utilizes MATLAB on a Windows 10 platform for the process of path planning. In order to setup up system for seamless wireless communication between the two PCs, ROS framework is implemented. The ROS message is of the type Int32. With reference to the PC on the robot, the corners are sent in a fixed order which is bottom left, bottom right, top right and top left. These u and v values are sent in the same order in the form of an $8n \times 1$ array, where n is the number of obstacles which has 4 corners per obstacle and each corner is described by a corresponding u and v value. The ROS message is published on the topic /obs_coor_bgs.

4.3.3 Path Planning using A* Algorithm

The A* algorithm for generating shortest paths is widely used to determine the least cost path to a node using Graph traverse approach. The approach to generate optimal paths is extensively studied [Koenig et al. (2004); Souissi et al. (2013)]. With a network being set up between the two PCs, MATLAB subscribes to the /obs_coor_bgs topic and retrieves the corner data. The $8n \times 1$ array is converted into a $4n \times 2$ matrix for simplicity and visualization. The obstacle corners which have been acquired from the ROS topic are utilized for the process of path planning using the

A* algorithm. Equation (4.2) gives the cost function $f(n)$ which is comprised of two metrics.

$$f(n) = g(n) + h(n) \tag{4.2}$$

where $g(n)$ is the total distance from the initial position to the current position. $h(n)$ is the heuristic function that is used to approximate distance from the current location to the goal state. It is considered a heuristic cost function $h(n)$ as metric to compute the relative cost of moving from the current node to the destination node [Mahadevi et al (2014)]. The more accurate the heuristic, the faster the goal state will be reached with much more accuracy. The steps involved in the functioning of the A* algorithm with respect to path planning have been listed below.

Steps in the A* algorithm

Step 1	Initialize all variables. [$g(n)$, $h(n)$, $f(n)$]
Step 2	Add the starting node (yet to visit list.)
Step 3	Define a stop condition to avoid an infinite loop.
Step 4	Define movement in terms of relative position.
Step 5	Look for the lowest f (cost function) on the starting node (becomes the current node).
Step 6	Check max iteration reached or not.
Step 7	Check if the current node is the same as target node.
Step 8	Check adjacent nodes to this current node to update the children node.
Step 9	If it is not movable or if it is on the "updated node", ignore it.
Step 10	Otherwise, create the new node with the parent node as the current node and update the position of the node.
Step 11	Stop when the target node added, in which case the path has been found.
Step 12	Fail to find the target node, and the adjacent nodes are empty-there is no path.

The 640×480 resolution pixel coordinates corresponding to the contour corners are scaled down to a 10×10 grid in order to reduce the computational power utilized by A*. A 10×10 matrix (MAP) is created which encodes different values to positions based on their characteristics. The rectangle corners in the MAP are considered as obstacles and the corresponding positions are assigned values of (-1). The start and the goal node are assigned values of 1 and 0 respectively. All the other nodes are considered as free space. The A* implementation runs and the path planner finds the shortest path avoiding all the grid locations where obstacles are present. The optimal path is then provided in terms of node locations from the 10×10 grid which can be used to traverse the shortest path from start to the goal location. The implementation of A* algorithm for the vision data captured from the camera is shown in Figure 4.8.

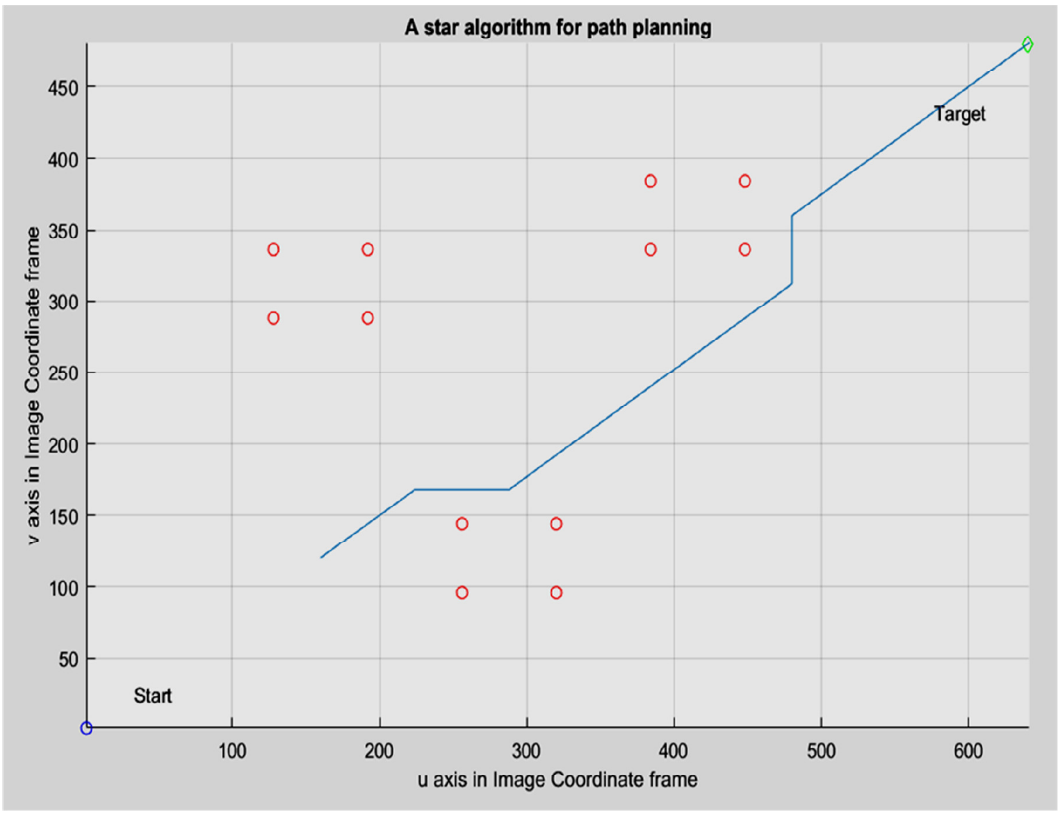

Figure 4.8: A* algorithm based path planning using vision-data

Algorithm workflow

In this sub-section, the entire workflow of the proposed mobile manipulation-based path planner is summarized using the proposed pseudo-code. The proposed pseudo-code discusses about various components of the mobile manipulation-based path planner which include object classification on basis of metric information, priority generation and multi-target motion cycles. Each of these components will be described in detail in the further sections.

```
SET GOAL
INIT MAP with all values as 2
INIT All_Object_corners_matrix
GET Number of Objects
FOR (Number of Objects*4) // four corners in a rectangular contour
        INPUT All_Object_corners_matrix
        SET Object_index to -1 in MAP
ENDFOR
// Split the object_corner_array into separate objects
INIT Cell Array Split_Objects, Object_Corners_Only, Filled_Objects, Manipulatable_Object_Corners
INIT Filled_Objects_Stacked_Matrix, Obj_Dimensions_Matrix, target_matrix
FOR i=1 till (Number of Obstacles)
        EXTRACT Split_Objects{i} for ith obstacle from All_Object_corners_matrix
        CALL Calculate_dimensions with Split_Objects{i} RETURNING WITH Width, Height,
manipulatable_flag, approach_flag, object index
        STORE Split_Objects{i} with manipulatable_flag, approach_flag, object index IN
Object_Corners_Only{i}
        STORE width, height with manipulatable_flag, approach_flag, object index IN
Obj_Dimensions_Matrix
        CALL Fill_Obstacle with Object_Corners_Only{i} RETURNING WITH Filled_Objects{i}
        UPDATE MAP with Filled_Objects{i}
        UPDATE Filled_Objects_Stacked_Matrix with Filled_Objects{i}
        IF manipulatable_flag == 1
                Number_of_Manipulatable_Objects ++
                Manipulatable_Object_Corners{i} = Object_Corners_Only{i}
        CALL Approach_Generator with Manipulatable_Object_Corners{i} RETURNING WITH
target_matrix
ENDIF
ENDFOR
INIT Priority_index, target_matrix_sorted, previous_coordinates
CALL Priority_Generator with target_matrix, Obj_Dimensions_Matrix RETURNING WITH
Priority_index, target_matrix_sorted
IF Priority_Index is empty
        DISPLAY ('No Obstacles to be picked')
ELSE
FOR i=1 till (Number_of_Manipulatable_Obstacle)

        // Pick up Object
        SET Start= (1,1)
        UPDATE MAP(Start)= 1
        SET Target= target_matrix_sorted(i)
UPDATE MAP(Target)= 0
        a_star_algorithm(Start, Target)
        STORE previous_coordinates with Start, Target
        draw_optimal path
        reinitalise map without obstacle(i)
        // Drop Object to Goal
        SET Start = previous_coordinates[Target]
        UPDATE MAP(Start)= 1
        SET Target = GOAL
UPDATE MAP(Target) = 0
        a_star_algorithm(Start, Target)
        draw_optimal path
ENDFOR
ENDIF
```

4.4 Mobile Manipulation-Based Path Planner

The focus of this research is to develop an algorithm for object cleanup which can be used for tasks like personal home cleaning, nuclear waste management and handling factory operations. The vision-based A* algorithm explained in the last section has been successful indetecting objects and plan a path in order to avoid objects. However, it suffers from certain fundamental limitations which prevent it from being used for the tasks of picking up objects using a mobile manipulation system.

4.4.1 Limitations of Vision-Based A* Algorithm

Firstly, the vision-based A* algorithm utilized for path planning is optimum for obstacle avoidance tasks. However, it is intuitive that the same algorithmic structure would not be suitable for developing a path planner with a core focus on manipulating handleable objects while avoiding objects which cannot be handled by the manipulator**.** Secondly, it considers different corners of the rectangular contours as separate point obstacles and not the four corners of the same object. It also does not capture the precise geometric extent of the object. As the four corners of the rectangular contours are not always closed objects, it may lead to an incorrect path passing through an object. Such a path will be incorrect as it will lead the robot to collide with the given obstacle.

4.4.2 Problem Definition

The central idea is to develop a mobile manipulation centric path planner for the purpose of multi-object clean-up operations. The robot path planner is provided a start location and a goal location. Depending upon the priority weights provided by the user for certain factors, the robot has to pick up the handleable objects and place them at the goal location. Here it is assumed that the mobile manipulator system has a defined feasible grasp range. Any grasps above this range are not feasible and not to be handled by the manipulator. However, the robot can pick up objects with dimensions less than the grasping range by reducing the size of the grasp. The mobile manipulation system does not have a bin or a temporary holding capacity to store the objects which have been lifted. Hence, the lifted objects need to be dropped off at the goal location before another object can be picked up. A dedicated return path is provided from the goal location to the start location so that the same mobile robot can carry out multi-target path planning and manipulation.

4.4.3 Object Classification on Basis of Metric Data

The individual object corner data extracted from each rectangular contour is stored as a separate entity which is associated with a particular object. Using the process of camera calibration, the image coordinates are converted into real-world coordinates. These real world-coordinates are utilized to find the dimensions of the object which are termed as width and height. The width and height are compared with the maximum feasible grasp size of the robotic manipulator. Depending upon this comparison, the object is labelled whether the object is handleable or not. A handleable object is provided with the flag value of 1, while a non-handleable object has a flag value of 0.

4.4.4 Generation

As the starting location and relative difference between the width and height is known, the object is assigned a direction of approach, the optimal dimension being the one which requires the minimal grasp size for the robotic manipulator. Hence, every object is assigned an approach flag. Approach flag of 1 refers to width approach, 2 refers to height approach and 0 refers to objects which cannot be grasped. Depending on the direction of approach, the object is provided a target value which is a suitable position for the mobile robot to approach and grasp the object from. A flowchart summarizing the object classification and approach generation on the basis of their metric dimensions is presented in Figure 4.9.

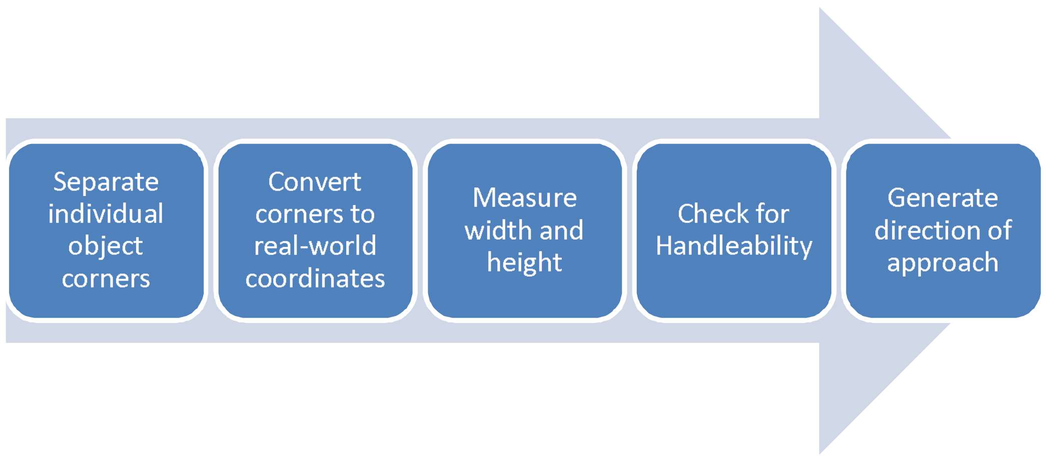
Separate individual object corners
Convert corners to real-world coordinates
Measure width and height
Check for Handleability
Generate direction of approach

Figure 4.9: Object classification and approach generation on basis of metric data

Pseudo-Code for Object Classification on basis of Handleability

```
SET robot_grasp_size
IF (WIDTH <robot_grasp_size) and (HEIGHT >robot_grasp_size)
        SET manipulatable_flag=1
        DISPLAY ('Object can be handled. Approach from Width side')
        SET approach_flag=1;    %Width approach
ELSE IF( (WIDTH >robot_grasp_size) and (HEIGHT <robot_grasp_size))
        SET manipulatable_flag=1
        DISPLAY ('Object can be handled. Approach from Height side')
        SET approach_flag=2    %height approach
ELSE IF( (WIDTH <robot_grasp_size) and (HEIGHT <robot_grasp_size))
        SET manipulatable_flag=1
IF(WIDTH < HEIGHT)
        DISPLAY ('Object can be handled. Approach from Width side')
        approach_flag=1    %Width approach
  ELSE IF (HEIGHT < WIDTH)
        DISPLAY('Object can be handled. Approach from Height side')
        approach_flag=2    %height approach
  ELSE IF(HEIGHT = WIDTH) %Width approach as default
        DISPLAY ('Object can be handled. Approach from Width side')
        approach_flag=1    %Width approach
ELSE IF (WIDTH >robot_grasp_size) and (HEIGHT >robot_grasp_size))
        SET manipulatable_flag=0
        DISPLAY ('Object cannot be handled')
        approach_flag=0       % Cannot be grasped
ENDIF
```

4.4.5 Priority Generation

If the map has one handleable object and one non-handleable object, there is no issue of priority as there is only one object to pick up and deposit to the goal location. However, when there are multiple handleable objects, the path planner needs to decide the order in which these objects are to be handled. The order of priority is generated by using a weighted cost function approach. This cost function uses three variables which are the normalized object area, the normalized distance from start to target (object location) and the normalized distance from the target (object locations) to the goal. These parameters are calculated for every object for calculating their individual weighted cost. The object with the lowest cost value is given the highest priority. The weighted cost function allows the path planner to adapt as per the user preferences. The normalized distance from start to the target value (object location) d_{st} is used to generate a heuristic indicating how far an object is from the start location. The normalized distance from start to target value is calculated using equation (4.3). The normalized distance from target (object location) to goal d_{tg} is used to generate a heuristic indicating how far an object is from the goal location. The normalized distance from target to goal value is calculated using equation (4.4).

$$d_{st} = \frac{\text{Dist. from start to current object (in cm)}}{\text{Dist. from start to object (in cm)}} \tag{4.3}$$

$$d_{tg} = \frac{\text{Dist. from current object to goal (in cm)}}{\text{Dist. from object to goal (in cm)}} \tag{4.4}$$

The normalized object area A_n is used to associate a dimensional size value with every object where n is the number of handleable objects. The normalised area of an object is computed using equation (4.5). The normalised values range from 0 to 1. All parameter values are normalised for obvious reason so that the terms are comparable otherwise it will lead to unbalanced parameters.

$$A_n = \frac{\text{Area of current object (in cm}^2\text{)}}{\text{Total area of all objects (in cm}^2\text{)}} \tag{4.5}$$

The weighted cost function c_n for each object can be calculated as shown in equation (4.6), where w_1, w_2 and w_3 are the weights given to A_n, d_{st} and d_{tg} respectively. The objects are sorted in an increasing order of weighted costs with the object with the lowest weight being given the highest priority. The weights range from 0 to 1, with 1 being the maximum weight given to one parameter and 0 being the minimum.

$$C_n = (w_1 \times d_{st}) + (w_2 \times d_{tg}) + (w_3 \times A_n) \tag{4.6}$$

4.4.6 Multi-Target Motion Cycles

After the handleable objects have been assigned a priority, the next step is to plan the shortest path from the start to target and then target to goal in order to carry the required objects to the goal location. The target values provided by the approach generator have been sorted as per the priority and will be utilized to execute the motion cycles. The actions taken in the two cycles have been termed as the manipulation and the goal cycle.

4.4.6.1 Manipulation cycle

Firstly, the Start location is set to a value of 1 and the Target location generated by the approach generator is set to a value 0 in the MAP matrix. With the given Start and Target location, A* algorithm is used to plan the optimal path to reach the target location. The previous start and target location are stored. The current target location will be utilized as the Start position for the goal cycle.

4.4.6.2 Goal cycle

Firstly, reinitialize the MAP after removing the current handleable object from the MAP matrix. Then, set the Start location as the previous target location from the manipulation cycle to a value of 1 and set the goal location to a value of 0 in the MAP matrix. With the given Start and Goal location, A* algorithm is used to plan the optimal path to reach the Goal location. The robot is assumed to return back to the initial start position via a permanent path. These cycles keep on repeating till all handleable objects present in the MAP are cleared.

4.5 Results and Discussion

The key step to improve the efficiency of the mobile manipulation-based path planner is classifying the objects as handleable or not. The algorithm also needs to generate the direction of approach. The first step in the experiment involves image data captured with the help of an overhead camera and then determining the dimensions of the object. The absolute error is computed for each object in order to obtain the variation from the ideal values for each object. The same object is placed at the extreme ends of the map as well on the center to check how the measurement error varies due to the change in the view. The maximum feasible robot grasp was assumed to be 20 cm for the selection of approach and handle ability. The objects and the corresponding detected contours from 5 different views are shown in Figure 4.10. The variation in measurement error for 5 different views is presented in Table 4.1.

Table 4.1: Variation of measurement error using overhead camera (Ideal dimensions: height = 35 cm, width = 15 cm)

Image No.	measured width (cm)	abs. error in width (cm)	measured height (cm)	abs. error in height (cm)	approach flag 1=width 2=height
1	13.1875	1.8125	31.3295	3.6705	1
2	13.6337	1.3663	31.3393	3.6607	1
3	13.6500	1.3500	32.7400	2.2600	1
4	15.2652	0.2652	33.2956	1.7044	1
5	13.7035	1.2965	32.5522	2.4478	1

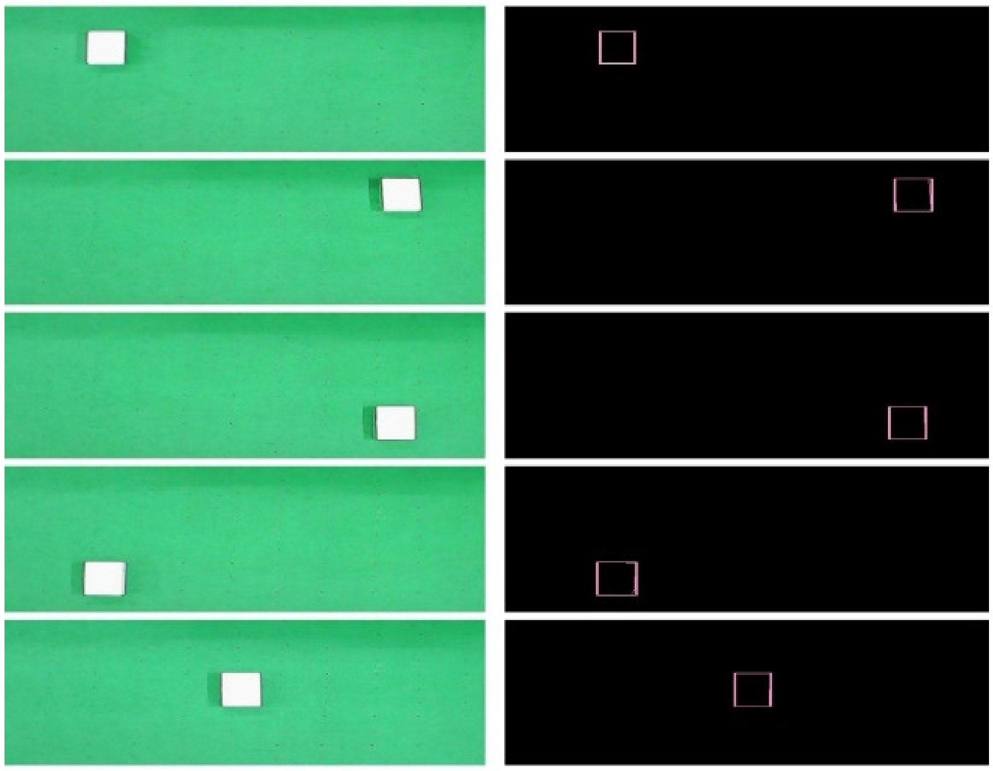

Figure 4.10: RGB images and detected rectangular contours for 5 different views

Error in measurement of objects from all different views is within ±4 cm in height and ±2 cm in width. Hence, this confirms that the overhead camera system can be used reliably to measure objects in the scene. The proposed algorithm has been implemented for a few cases to generalize the application.

In 2 non-handleable objects and 1 handleable object scenario, there is 1 handleable object and there are 2 non-handleable objects which act as obstacles and have to be avoided by the mobile manipulation-based path planner. It is assumed that the robot's maximum feasible grasp size is 20 cm. The metric data, as well as handleable with approach flag for two non-handleable objects and one handleable object case, is presented in Table 4.2.

Table 4.2: Two non-handleable objects and one handleable object scenario

Object No.	**width (cm)**	**height (cm)**	**handleable flag 0 = non-handleable 1= handleable**	**approach flag 0= non-handleable 1=width Side 2=height side**
1	31.8757	99.4239	0	0
2	21.7037	19.5875	1	2
3	32.8663	75.1148	0	0

The handleable object is classified on basis of the simulated metric data. The manipulation and goal cycle are demonstrated and the handleable object is picked up and placed at the goal location. The manipulation cycle and the goal cycle for carrying the handleable object are shown in Figures. 4.11 to 4.12.

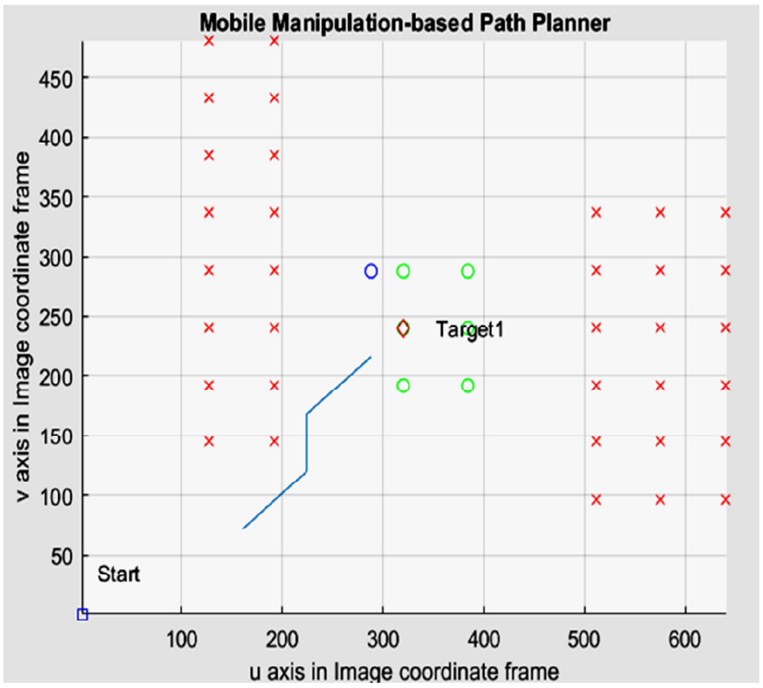

Figure 4.11: Manipulation cycle for handleable object

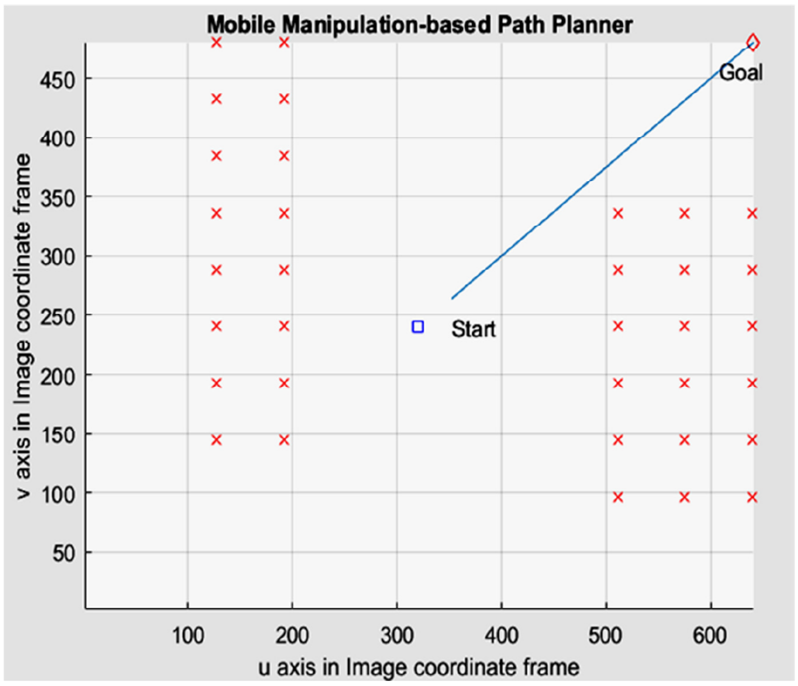

Figure 4.12: Goal cycle for handleable object

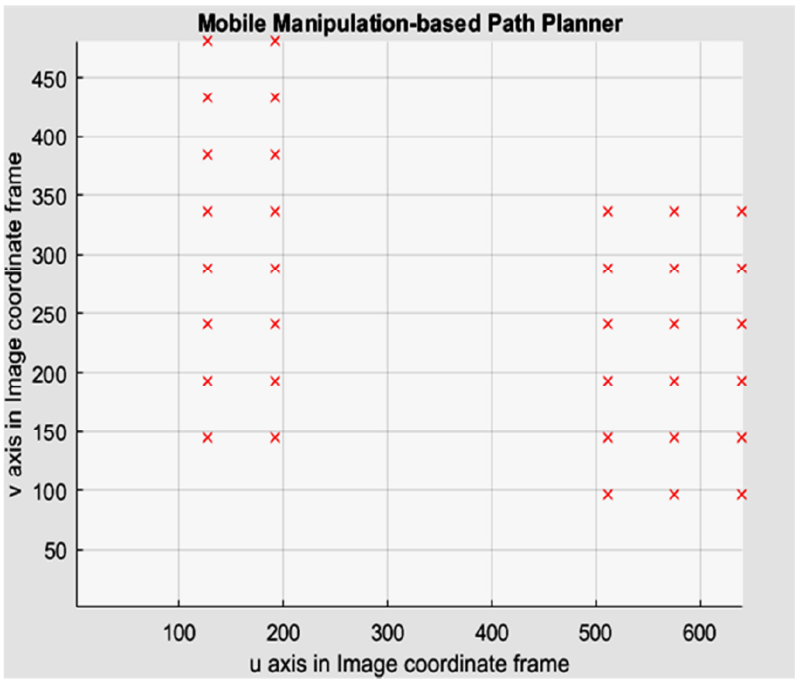

Figure 4.13: Non-handleable objects remain as obstacles

The result of the work demonstrates the behavior of the algorithm for multi-object pickup operations. In this work, the objects are classified into handleable objects and non-handleable objects based on the simulated metric data. To demonstrate the implementation, the manipulation and goal cycles for multi-target object clean-up have been simulated. In this case the priority generation and priority costs have been tabulated for different weights/priorities. The total number of nodes explored and total path cost over all the cycles of object clean-up is used as a performance metric to evaluate it.

Table 4.3: Dimensions of multiple handleable and non-handleable objects scenario

Object no.	**width (cm)**	**height (cm)**	**area (cm^2)**	**Handleable flag 0 = non-handleable 1= handleable**
1	21.0007	19.4747	408.9823	1
2	25.1228	18.6549	468.6633	1
3	31.8031	36.1617	1150.01	0
4	22.7836	19.4540	443.2322	1
5	34.9219	37.4431	1307.6	1

In Multiple Handleable and Non-Handleable Objects Scenario, the map has multiple handleable objects as well as multiple non-handleable objects. The weights set for the priority generator becomes a deciding factor regarding which object will be chosen first. The dimensions of the five objects in the Map are shown in Table 4.3.

Tthe development of a mobile manipulation-based path planner was discussed in next page, and the performance of the path planner was tested by placing different sets of weights in the priority generator.

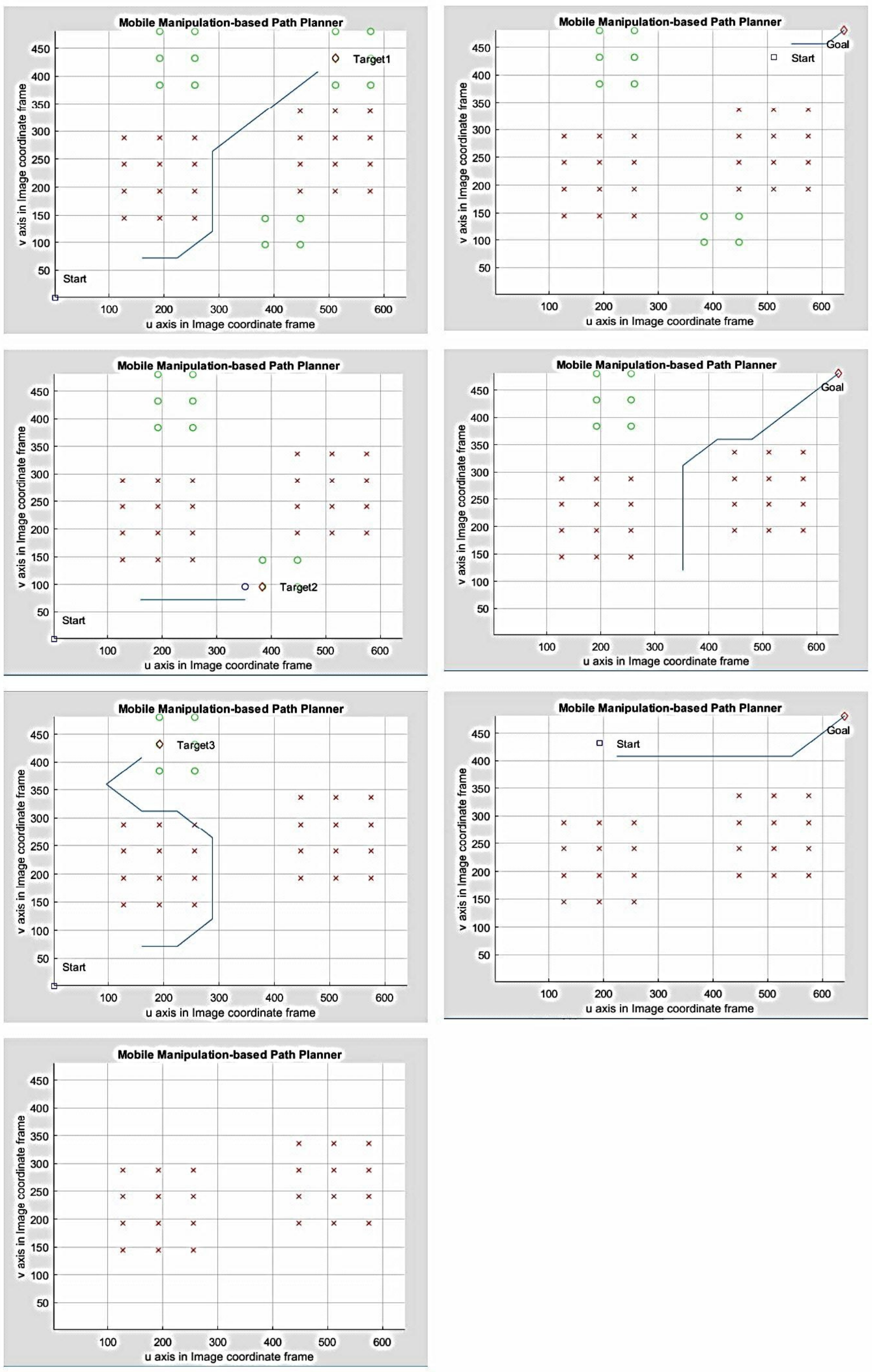

Figure 4.14: Path planner for Case 1 with weight 1= 0.33, weight 2= 0.33, weight 3= 0.33

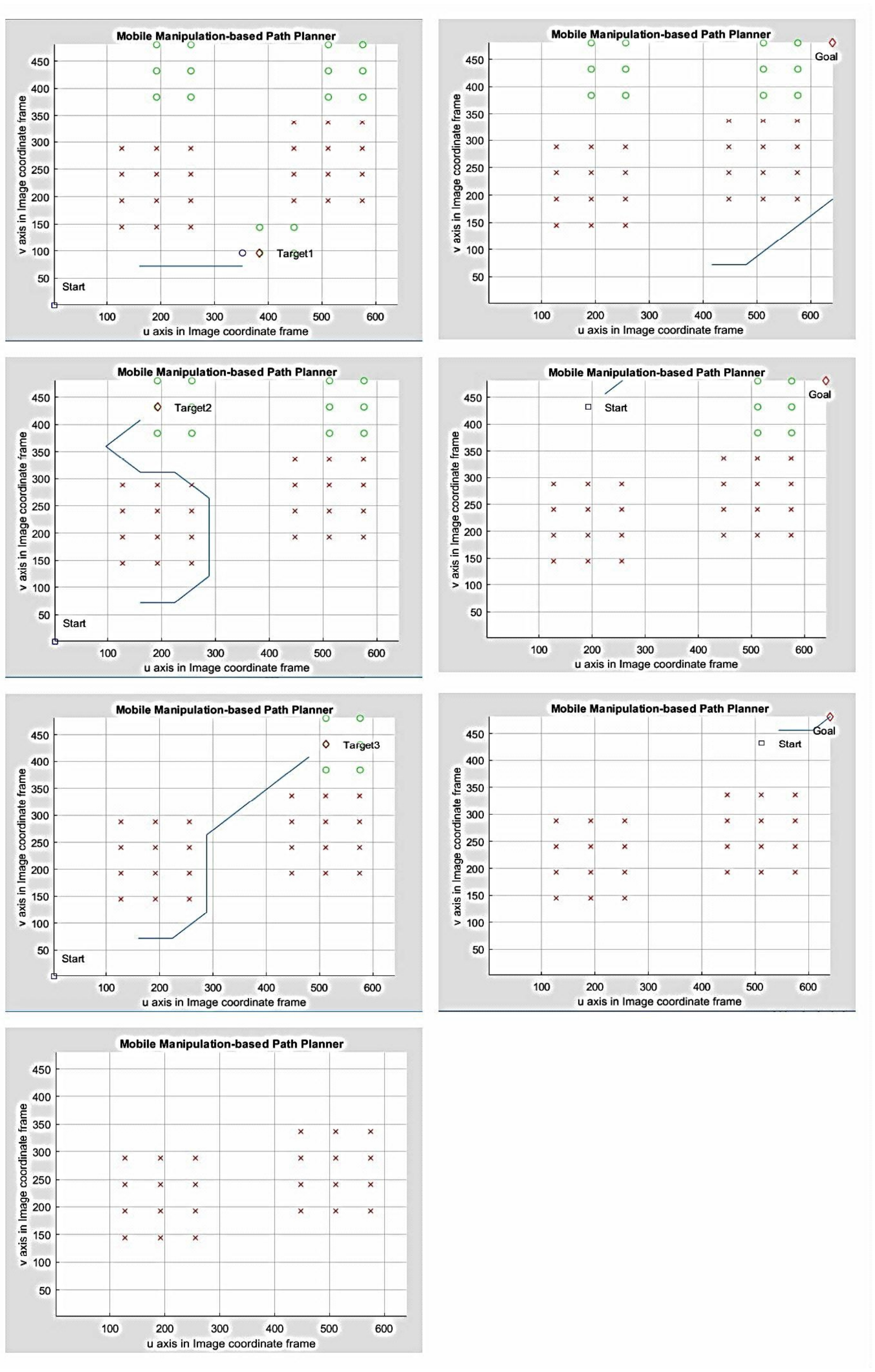

Figure 4.15: Path planner for Case 2 with weight 1= 0.8, weight 2= 0.1, weight 3= 0.1

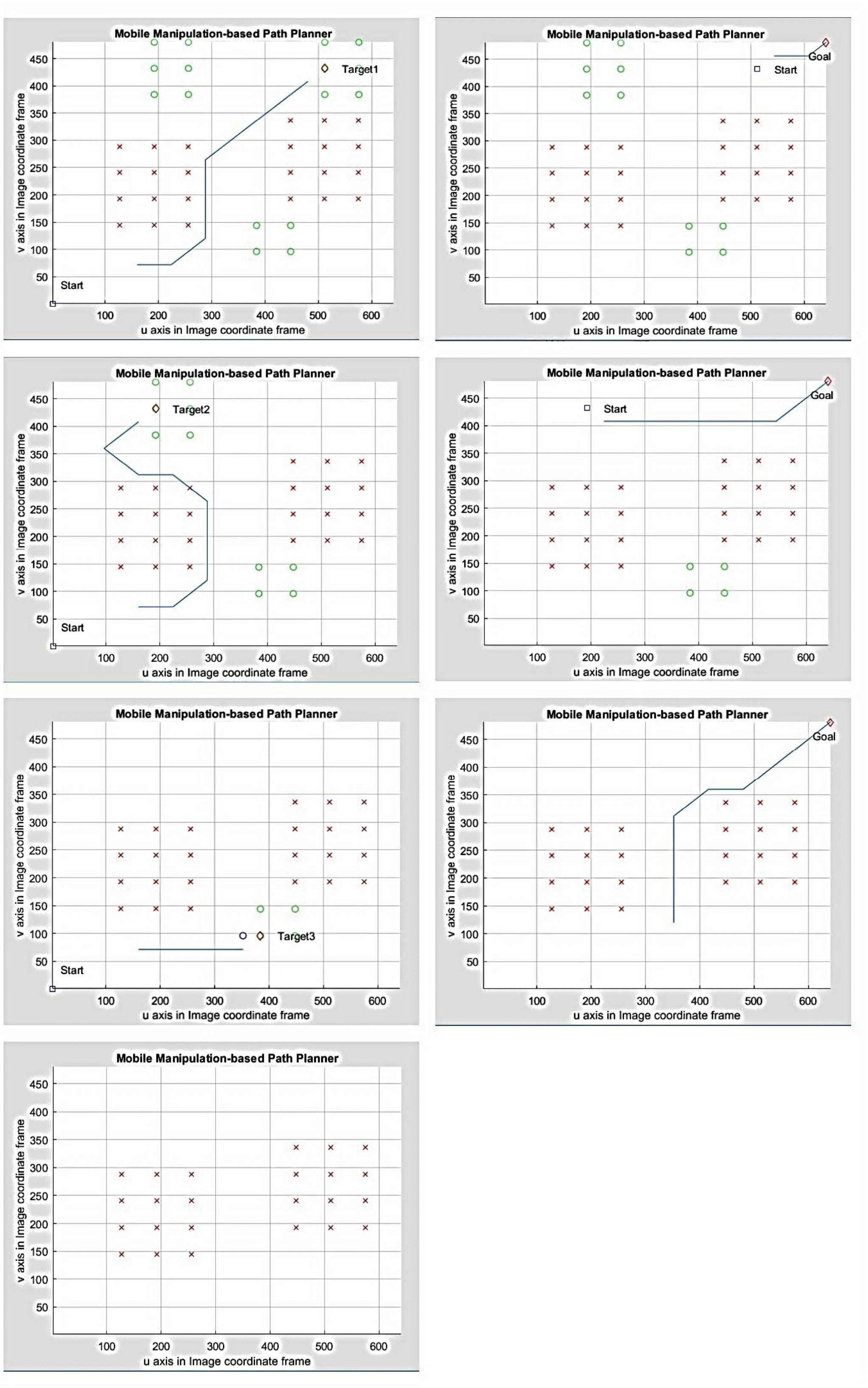

Figure 4.16: Path planner for Case 3 with weight 1= 0.1, weight 2= 0.8, weight 3= 0.1

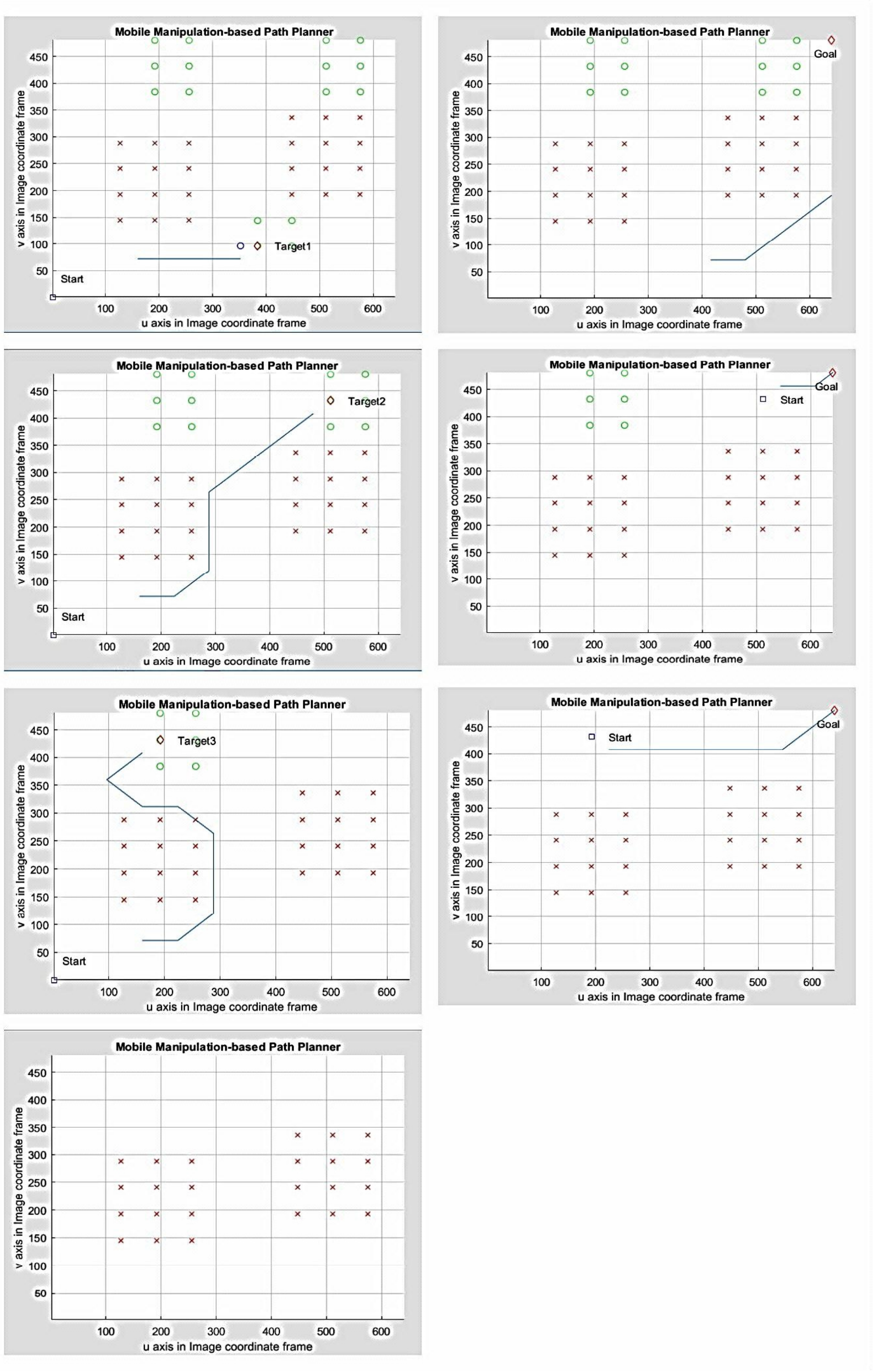

Figure 4.17: Path planner for Case 4 with weight 1= 0.1, weight 2= 0.8, weight 3= 0.1

Table 4.4: Performance of the path planner with change in priority weights

Case no.	weight (w_1, w_2, w_3)			priority costs for handleable objects (object no.)			total number of nodes explored for all cycles	total path cost for all cycles
				obj. 1	obj. 2	obj. 3		
1	0.33	0.33	0.33	0.3258	0.3535	0.3108	151	50.6274
2	0.8	0.1	0.1	0.2476	0.3479	0.4045	152	51.2132
3	0.1	0.8	0.1	0.4241	0.3677	0.2081	147	50.6274
4	0.1	0.1	0.8	0.3155	0.3555	0.3291	161	51.2132

Table 4.4 shows to how the performance of the algorithm varies with a change in the weights, keeping the object positions to be stationary.

Figure 4.14 presents the path planner for Case 1 with weight 1 w_1 = 0.33, weight 2 w_2= 0.33, weight 3 w_3 = 0.33 where total number of nodes of 151 explored for all cycles with total path cost 50.6274. Likewise Figures 4.15, 4.16, 4.17 present the path planner for Case 2, Case 3, Case 4 respectively. Figures 4.14 to 4.17 summarize the full workflow for mobile manipulation for Case no. 1, 2, 3, 4 respectively.

It can be observed that the algorithm performs the best for weight values of 0.1, 0.8 and 0.1 respectively. This refers to the condition where the distance from the target to goal for each object is assigned highest weight. This leads to a logical conclusion that ideally, a heavier weight should be given to d_{tg} as compared to d_{st} as the distance form target to goal. The explanation being that the previous object pick-up tasks would clear up the path and allow a less cluttered path to navigate with the heavier object. The full operation for object clean-up for multiple handleable and non-handleable objects is shown in Figure 4.16 for Case 3 from Table 4.4 where total number of nodes of 147 explored for all cycles with total path cost 50.6274 for the process of multi-target path planning, which was lowest among all the other combination of weights.

The objects were classified on basis of their metric data from the images provided by the overhead camera. The purpose is to reduce the errors from centimeter to millimeter level accuracy in object measurement by utilizing a local object measurement scheme.

4.6 Conclusion

In this work, a new approach to robot path planning for mobile manipulators has been proposed. Here, a vision-based A* algorithm for mobile robot path planning was developed. An overhead camera was utilized to capture images. These images were pre-processed to isolate the obstacles in the scene using background subtraction. Rectangular-based contours were utilized for extracting corners of the required obstacles. A vision-based path planning approach was implemented using A* algorithm in order to avoid the obstacles and reach the goal location using the shortest path. A mobile manipulation-based path planner was developed with a focus on multi-target object clean-up operations. The algorithm was developed with a fundamental idea of classifying objects as handleable or non-handleable from real-time measurements. A weighted cost function approach was utilized for priority generation in case of multi-object cleanup operations. The efficacy of different parts of the algorithm was tested in a series of experiments. Firstly, the process of object measurement was done by keeping the same object in various views in order to understand the variation in the error. The error of the measurement process was found out to be in a range of ± 4cm in height and ± 2cm in width. Secondly, the entire workflow of the mobile manipulation-based path planner was demonstrated using a single handleable object and two non-handleable object scenarios. This result also showcased that objects were correctly classified as handleable and non-handleable from the object metric data.

Thirdly, the process of priority generation was tested by changing the weights of the algorithm and analyzing the performance of the path planner. In this case, the path planner with weights of 0.8 for d_{tg} 0.1 for d_{st} and 0.1 for area of the object parameters A_n resulted in optimum performance. This combination of weights required a total of 147 nodes from the search space to be explored and had a total path cost of 50.6274 for the process of multi-target path planning, which was lowest among all the other combination of weights. The future scope of work includes a comparison of probabilistic approaches like Random Rapidly-Exploring Trees (RRTs) and Probabilistic Cell Decomposition (PCD) with the A* algorithm after integration with the proposed mobile manipulation-based path planner. The performance of this path-planner will be tested in real-time on a two-wheeled differential robot platform for further evaluation and analysis.