RWTH AACHEN UNIVERSITY

CONSTRUCTION & ROBOTICS MASTER'S PROGRAM

2024 SS PROTOTYPING PROJECT

Group B

ATOM ANT – INNOK HEROS 223
PROTOTYPING PROJECT

JAMAL SIDI , ZEIAD AHMED , YAGIZ ERAY ESGIN

1. Introduction

The construction sector is facing several significant challenges, including a lack of workers and higher safety hazards. An aging workforce and a decrease in the number of new workers contribute to the labor crisis, which makes automated solutions more and more important for tasks that were previously completed by people [1]. Furthermore, the demand for safer work conditions is essential because the construction industry has among the highest rates of occupational fatalities and injuries [2]. By automating dangerous and repetitive processes, robotics offers feasible alternatives that improve overall productivity and safety while reducing the risk of accidents and injuries. In addition, robots are highly precise and consistent in their work, and there are fewer errors and rework demands, which results in more efficient operations and better project outcomes.

One of the important challenges in construction sites are material transportation and the reliance on manual labor. Current methods of material handling, such as using wheelbarrows or forklifts, are inefficient, costly, and prone to safety risks. These inefficiencies drive up project costs, extend timelines, and can lead to safety concerns due to heavy reliance on human labor [3].

The primary goals of the client are to optimize material handling processes through increased overall effectiveness and quality of operations, cost reduction, and safety improvements. By upgrading building methods, the project seeks to preserve a competitive edge through the utilization of innovative equipment such as the INNOK HEROS 223. The main objectives are to maximize output, reduce waste, effectively fulfill project deadlines, and maximize resource use. For the robot to function effectively in dynamic construction contexts and offer flexibility in integration and adaptation, the workflow the project offers is quite important.

The advantages of automation are evident when one contrasts robotic solutions with traditional material handling methods. Although robotic material handling systems, like the INNOK HEROS 223, have a higher initial cost, they save a lot of money over time in labor, maintenance, and costs related to accidents. Traditional methods involve continuous expenses, like personnel and forklift maintenance, whereas robotic solutions significantly lower these prices and the Atom Ant Innok Robot becomes a more valuable product in terms of the overall budget in construction sites.

The prototyping project focuses on leveraging advanced robotics to automate material transportation. By integrating the INNOK HEROS 223 robot, we aim to demonstrate how automation can streamline transportation operations inside the construction site, reduce manual labor risks, and ensure higher precision and consistency in construction tasks [4]. This project will showcase how robotic systems can transform construction sites by optimizing efficiency, safety, and overall project outcomes.

2. Concept Project

The primary goal of this project is to implement an automated solution for material transfer within construction sites. The robot selected for this task is the **INNOK HEROS 223**, a versatile and customizable robotic capable of transporting materials.

Fig.01 Innok Heros 223 Material Handling

2.1 The Current Process in the Construction site

The process of transferring materials from the storage area to the construction zone on the construction site mostly involves using a combination of manual work and automated assistance. For short distances or in small locations where heavier machinery is difficult to operate, workers transfer materials using wheelbarrows and manual handling techniques, which are still widely used in the process. Motorized wheelbarrows are occasionally used for longer transfers and bigger loads; they provide extra assistance and reduce muscular strain.

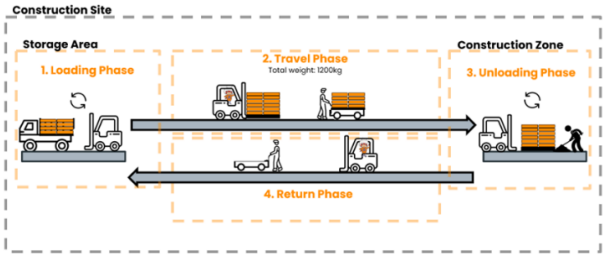


Fig.02 The Current Process in the Construction site

Larger or palletized products that need to be moved horizontally throughout the site are more often moved using forklifts. These materials are taken from the storage area by the forklift during the loading step. The load is moved to the construction zone by the forklift as it makes its way across the site during the travel phase. When it arrives, the forklift unloads the supplies so that construction workers can start using them right away. The forklift then heads back to the storage area in preparation for the next load. Although mechanical equipment such as wheelbarrows and forklifts are utilized to increase productivity, physical labor is still an essential aspect of the material handling process, especially in places where mechanized equipment is not practical. The efficient and safe transportation of materials throughout the site is ensured by this combined strategy.

2.2 Case Study - Construction Site

The Process of a Completed Material Transfer Mission

1. Retrieve Material from Storage: The location of the necessary materials has been programmed into the INNOK HEROS 223 robot. It navigates independently to the storage place.

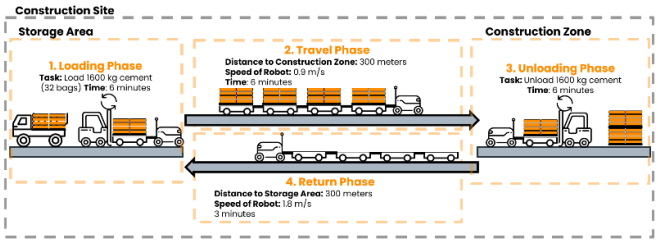


Fig.03 The Process of a Completed Material

2. Load Material onto Robot: Materials are securely placed in its platform by labor or forklift and adjusted their grasp as needed [3].

3. Transport Material to Work Zone: The robot autonomously follows the path that was calculated before and reach the aim point.

4. Unload Material at Work Zone: Upon reaching the work zone, the robot positions itself for unloading. Lastly, Labor or forklift precisely place the materials and verify correct placement.

Comparison with the current transportation process

The comparison illustrates the efficiency of the INNOK HEROS 223 robot versus the current manual labor process in material transportation on construction sites.

1.Trips per Hour

Total Time per Trip: 21 minutes
Trips per Hour: 60 minutes/21 minutes ≈ **3 trips/hour**

2.Operational Hours per Day (Battery Life): **16 hours/day**

3.Total Trips per Day: 16 hours/day × 3 trips/hour ≈ **48 trips/day**

4.Total Weight Transported per Day:

1600 kg x **48**

= **76800** kg/day

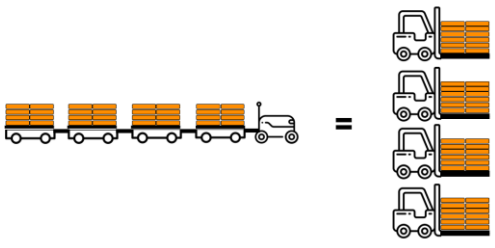
The current process in the construction industry

1. **Average Load per Trip:** 1200 kg
2. **Trips per Hour:** 2 trips/hour
3. **Operational Hours per Day:** 8 hours
4. **Total Weight Transported per Day:** 8 hours/day×2 trip/hour×1200 kg/trip=**19200 kg/day**

Efficiency Gain

1. **Innok Heros 223:** 76,800 kg/day
2. **Manual Labor:** 19200 kg/day
3. **Efficiency Improvement:** 76,800/19,200

= **4 X more efficient**



The comparison shows that the INNOK HEROS 223 robot is significantly more efficient than the current manual labor process on construction sites. The robot can operate for longer hours and complete more trips per day, leading to a substantial increase in the total weight transported. As a result, the robot offers a fourfold improvement in efficiency compared to manual labor, making it a much more effective solution for material handling tasks.

2.3 The Workflow of the project

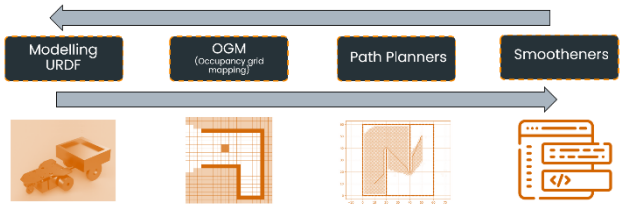


Fig.04 The Workflow of the project

2.4 OGM (Occupancy grid mapping)

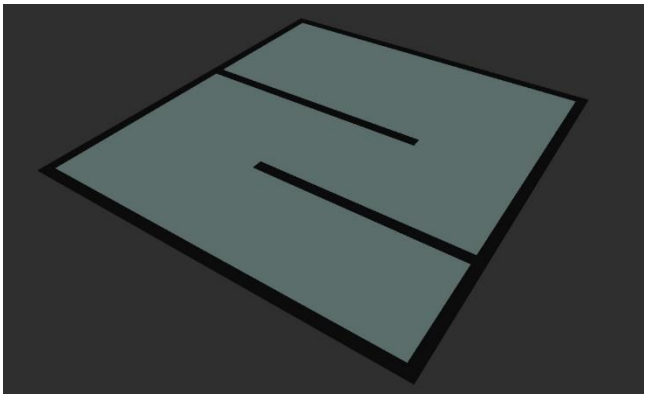


Fig.05 OGM (Occupancy grid mapping)

2.5 Path Planners

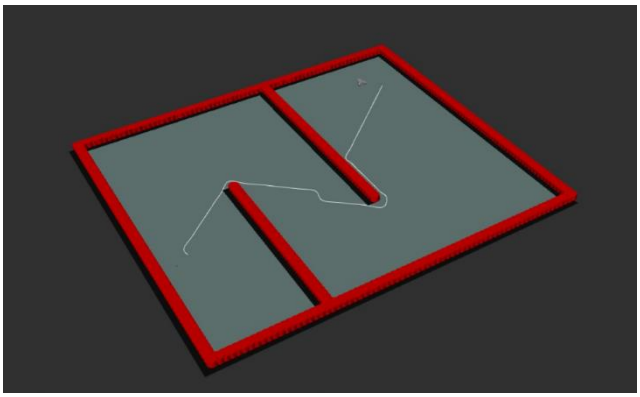


Fig.06 Path Planners

2.6 Smootheners

For Smoothing the lines of the path planner, different methods were compared. Cubic spline vs Beziar curvature: The main difference between Bezier and cubic curves and splines is that with a Bezier curve the two of the control points form the end points of the curve and the remaining control points are off the curve. With a cubic spline, all the control points are on the curve [6][7].the Cubic spline method was chosen after careful consideration.

A cubic spline is a smooth curve that is created by piecewise third-order polynomials linking a set of control points. The curve has a smooth and natural-looking route because each segment is specified by a unique cubic polynomial, which guarantees the curve passes through all of the control points with continuous first and second derivatives.

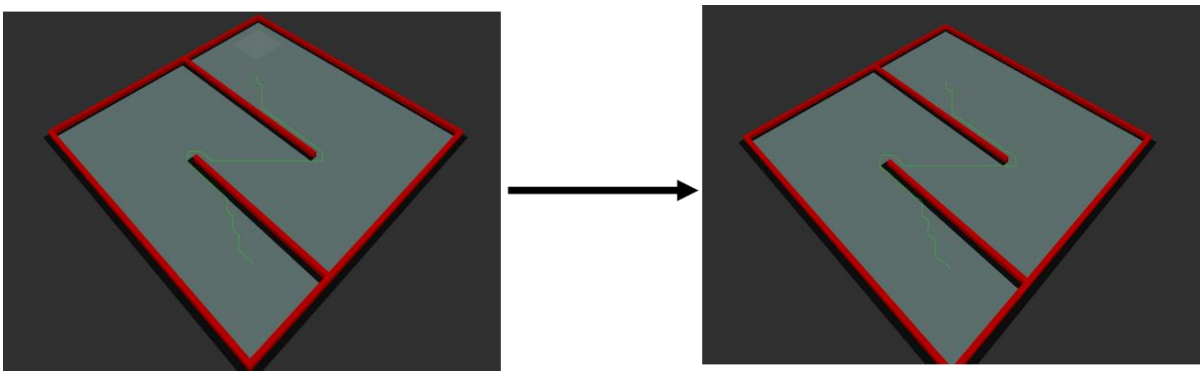


Fig.07 Smootheners

2.7 URDF | Model

To speed up the process of turning 3D models from Autodesk Fusion 360 into URDF (Unified Robot Description Format) files—which are required for robot simulation in ROS 2—we used the fusion2urdf-ros2 tool in our project[8]. Our robot model, complete with joints and linkages, could be exported straight from Fusion 360 thanks to the script, which also guaranteed simulation setup accuracy and saved time. We could go from design to simulation fast with this tool, successfully integrating our robotic system into ROS 2 for testing and validation.

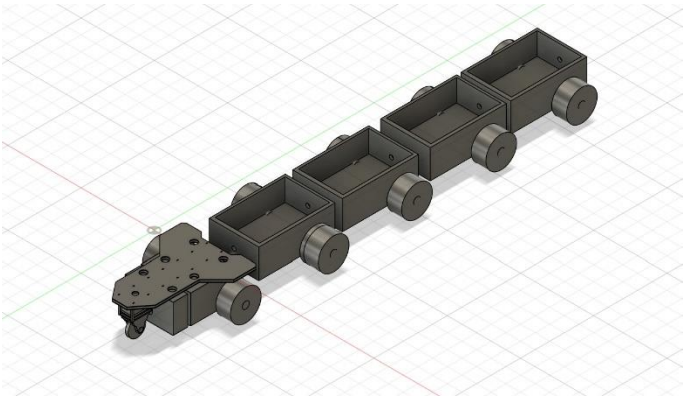


Fig.08 URDF | Model.Fusion360

In order to ensure correct hierarchy and joint definitions, we organized our CAD model in Fusion 360 in accordance with the tool's specifications during the process. Each of the required URDF files, such as the.stl files for visualization and the.launch.py scripts for simple deployment in simulation environments, were generated by the script upon execution.

3. System Architecture

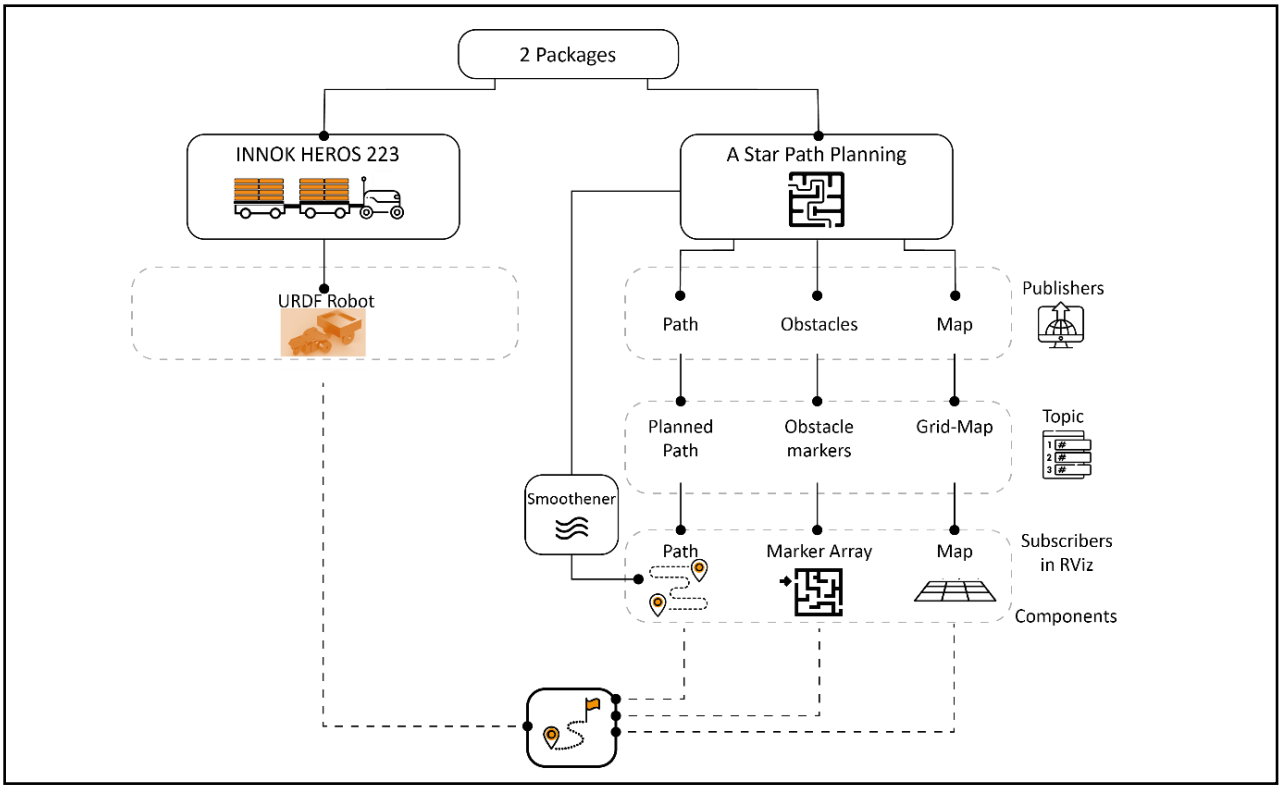


Diagram01 System Architecture

3.2 Explanation of the Code

Innok Heros Launch File

1. Import Required Modules

Before defining the function, you need to import the required modules to use their functionalities. This would typically look like:

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch.conditions import IfCondition, UnlessCondition
import xacro
```

2. Define the generate_launch_description Function

The function generate_launch_description is the main entry point for defining the ROS 2 launch description. It describes the nodes and configurations that will be launched.

```
def generate_launch_description():
```

3. Getting Package Share Directory

Retrieve the path to the package directory for innok_description:

```
share_dir = get_package_share_directory('innok_description')
```

4. Processing the Xacro File

Combine the path to the Xacro file and process it:

```
xacro_file = os.path.join(share_dir, 'urdf', 'innok.xacro')
robot_description_config = xacro.process_file(xacro_file)
robot_urdf = robot_description_config.toxml()
```

xacro_file builds the full path to the Xacro file.

robot_description_config processes the Xacro file to generate a robot description.

robot_urdf converts the processed Xacro configuration to XML format.

5. Declaring Launch Arguments

Define a launch argument for whether to show the GUI:

```
gui_arg = DeclareLaunchArgument(
    name='gui',
    default_value='True'
)
```

name='gui' defines the argument name.

default_value='True' sets the default value if not overridden.

6. Configuring Launch Conditions

Create a launch configuration to read the argument:

```
show_gui = LaunchConfiguration('gui')
```

7. Defining Nodes

Robot State Publisher Node

Publish robot state information:

```
robot_state_publisher_node = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    name='robot_state_publisher',
    parameters=[
        {'robot_description': robot_urdf}
    ]
)
```

package='robot_state_publisher' specifies the ROS 2 package.

executable='robot_state_publisher' specifies the node executable.

parameters pass the robot description to the node.

Joint State Publisher Node

Publish joint states, conditionally based on the gui argument:

```
joint_state_publisher_node = Node(
    condition=UnlessCondition(show_gui),
    package='joint_state_publisher',
    executable='joint_state_publisher',
    name='joint_state_publisher'
)
```

- condition=UnlessCondition(show_gui) ensures this node runs only if gui is False.

RViz Node

Launch RViz for visualization:

```
rviz_node = Node(
    package='rviz2',
    executable='rviz2',
    name='rviz2',
    arguments=['-d', rviz_config_file],
    output='screen'
)
```

- package='rviz2' specifies the RViz package.
- executable='rviz2' specifies the RViz executable.
- arguments=['-d', rviz_config_file] sets the configuration file for RViz.

8. Returning Launch Description

Assemble and return the launch description:

```
return LaunchDescription([
    gui_arg,
    robot_state_publisher_node,
    joint_state_publisher_node,
    joint_state_publisher_gui_node,
    rviz_node
])
```

This function returns a LaunchDescription object containing all defined launch configurations, nodes, and arguments.

Path planning python file

1. Node Initialization

The **AStarPlannerNode** class initializes the ROS node, sets up publishers and subscribers, and prepares the grid for path planning:

```
class AStarPlannerNode(Node):

    def __init__(self):
        super().__init__('a_star_planner')
```

```
        self.get_logger().info('AStarPlannerNode has been started')

        # ROS publishers
        self.path_pub = self.create_publisher(Path, 'planned_path',
10)
        self.marker_pub = self.create_publisher(MarkerArray,
'obstacle_markers', 10)
        self.grid_pub = self.create_publisher(OccupancyGrid,
'grid_map', 10)

        # ROS subscribers
        self.create_subscription(PointStamped, 'clicked_point',
self.clicked_point_callback, 10)

        # Start and goal points
        self.start_point = None
        self.goal_point = None

        # Default obstacle positions
        self.ox, self.oy = self.default_obstacles()

        # Initialize the grid map at the start
        self.publish_grid_map(self.ox, self.oy, 2.0,
round(min(self.ox)), round(min(self.oy)), 35, 35)
```

- Node Initialization: This block initializes the ROS 2 node with the name 'a_star_planner'.
- ROS Publishers and Subscribers: It creates publishers for the planned path (Path), obstacle markers (MarkerArray), and occupancy grid map (OccupancyGrid). A subscriber listens for points clicked by the user, which are used as start and goal points.
- Obstacle Initialization: Default obstacles are created using the default_obstacles method.
- Grid Map Initialization: The initial grid map is published using the publish_grid_map method.

2. Handling User Input

```
def clicked_point_callback(self, msg: PointStamped):
    if self.start_point is None:
        self.start_point = (msg.point.x, msg.point.y)
        self.get_logger().info(f'Updated start point to
{self.start_point}')
    else:
        self.goal_point = (msg.point.x, msg.point.y)
        self.get_logger().info(f'Updated goal point to
{self.goal_point}')
        self.plan_and_publish_path(self.start_point[0],
self.start_point[1],
                                   self.goal_point[0],
self.goal_point[1],
                                   self.ox, self.oy, 2.0, 1.0)
```

```
        self.start_point = self.goal_point  # Update start point to
the last goal point
```

Point Callback: This method is triggered when a point is clicked in the visualization tool. The first clicked point is set as the start point, and the second is set as the goal point. Once both points are set, the A* path planning function (plan_and_publish_path) is called.

3. Defining Obstacles

```
def default_obstacles(self):
    ox, oy = [], []
    # Defines boundaries and obstacles within the grid
    for i in range(-10, 60):
        ox.append(i)
        oy.append(-10.0)
    for i in range(-10, 60):
        ox.append(60.0)
        oy.append(i)
    for i in range(-10, 61):
        ox.append(i)
        oy.append(60.0)
    for i in range(-10, 61):
        ox.append(-10.0)
        oy.append(i)
    for i in range(-10, 40):
        ox.append(20.0)
        oy.append(i)
    for i in range(0, 40):
        ox.append(40.0)
        oy.append(60.0 - i)
    return ox, oy
```

Obstacle Generation: This method generates default obstacle positions on the grid, forming boundaries and additional barriers within the grid. These are represented by the ox (obstacle x-coordinates) and oy (obstacle y-coordinates) lists.

4. Path Planning with A*

```
def plan_and_publish_path(self, sx, sy, gx, gy, ox, oy, resolution,
rr):
    self.get_logger().info('Starting path planning...')
```

*A Path Planning Method**: This method uses the A* algorithm to find the optimal path from the start (sx, sy) to the goal (gx, gy).

Helper Functions within A* Method

```
def calc_heuristic(n1, n2):
    w = 1.0  # weight of heuristic
    d = w * math.hypot(n1[0] - n2[0], n1[1] - n2[1])
    return d
```

Heuristic Calculation: This function computes the heuristic distance (Euclidean distance) between two nodes, n1 and n2, weighted by w.

```
def calc_grid_position(index, min_position):
    return index * resolution + min_position
```

Grid Position Calculation: Converts an index on the grid to a coordinate position using the resolution of the grid.

```
def calc_xy_index(position, min_pos):
    return round((position - min_pos) / resolution)
```

Index Calculation: Converts a position to a grid index for efficient grid-based calculations.

```
def calc_grid_index(node):
    return (node[1] - min_y) * x_width + (node[0] - min_x)
```

Node Index Calculation: Calculates a unique index for a grid node based on its x and y positions.

```
def verify_node(node):
    px = calc_grid_position(node[0], min_x)
    py = calc_grid_position(node[1], min_y)

    if px < min_x or py < min_y or px >= max_x or py >= max_y:
        return False

    # collision check
    for iox, ioy in zip(ox, oy):
        d = math.hypot(iox - px, ioy - py)
        if d <= rr:
            return False
    return True
```

Node Verification: Checks if a node is within the boundaries and does not collide with any obstacles.

Motion Model

```
def get_motion_model():
    # dx, dy, cost
    motion = [[1, 0, 1],
              [0, 1, 1],
              [-1, 0, 1],
              [0, -1, 1],
              [-1, -1, math.sqrt(2)],
              [-1, 1, math.sqrt(2)],
              [1, -1, math.sqrt(2)],
              [1, 1, math.sqrt(2)]]
    return motion
```

Motion Model: Defines the possible movements from a given node to its neighbors and their respective costs. This includes diagonal movements with a cost of sqrt(2).

Core A* Loop

The A* search algorithm works by exploring the grid from the start node, using a priority queue (implemented with a dictionary) to select the most promising nodes based on the cost and heuristic.

```
while True:
    if len(open_set) == 0:
        self.get_logger().info("Open set is empty..")
        break

    c_id = min(
        open_set,
        key=lambda o: open_set[o][2] + calc_heuristic(goal_node,
open_set[o]))
    current = open_set[c_id]

    if current[0] == goal_node[0] and current[1] == goal_node[1]:
        self.get_logger().info("Goal found")
        goal_node = current
        break

    del open_set[c_id]
    closed_set[c_id] = current

    for i, _ in enumerate(motion):
        node = (current[0] + motion[i][0],
                current[1] + motion[i][1],
                current[2] + motion[i][2], c_id)
        n_id = calc_grid_index(node)

        if not verify_node(node):
            continue

        if n_id in closed_set:
            continue

        if n_id not in open_set:
            open_set[n_id] = node
        else:
            if open_set[n_id][2] > node[2]:
                open_set[n_id] = node
```

Open and Closed Sets: open_set keeps track of nodes to be explored, while closed_set keeps track of nodes already explored.

Node Selection: The node with the minimum total cost (cost from start + heuristic to goal) is selected from the open_set.

Goal Check: If the selected node is the goal, the search ends.

Neighbor Exploration: For each possible motion, the algorithm checks if the neighbor node is valid (inside boundaries, not colliding with obstacles) and updates it in the open_set if it's a better path.

5. Path Reconstruction and Smoothing

```
rx, ry = [], []
parent_index = goal_node[3]
while parent_index != -1:
    n = closed_set[parent_index]
    rx.append(calc_grid_position(n[0], min_x))
    ry.append(calc_grid_position(n[1], min_y))
    parent_index = n[3]

self.get_logger().info(f"Path
```

Summary

This Python script is designed to control a robot's movement by planning the best path from a starting point to a goal in an environment filled with obstacles. It uses the A* algorithm, a popular method for finding the shortest path on a grid while avoiding collisions with obstacles.

The node listens for user input to set the start and goal points, automatically plans a path when both points are set, and publishes this path for the robot to follow. It also manages the visualization of obstacles and the grid map, smoothing the computed path to make the robot's movements more natural.

Key components include determining obstacle positions, calculating the best route to avoid them, and continuously updating the path as needed. The script runs within a ROS (Robot Operating System) environment and interacts with other components via message passing.

4. Result and Benefits

The use of the INNOK HEROS 223 robot for material handling tasks on building sites indicates an important advancement in project management, efficiency, and safety. The robot improves precision and reduces cost of labor and material handling time by automating labor-intensive and repetitive operations. This results in faster project completion, higher-quality outputs with less rework, and better results overall. Because of its capacity to reduce risks in hazardous environments, it enhances safety. Furthermore, the robot's adaptability and scalability across a range of project sizes and settings make it a flexible solution that meets a variety of operational requirements. In addition, by creating the framework for future robotics and AI integration in the construction industry, this project prepares businesses to obtain a competitive advantage in a sector that is becoming more and more driven by technological innovation.

The INNOK HEROS 223 robot's code implementation is crucial for the project's success since it allows accurate and effective material handling in dynamic constructing limitations. The program includes fundamental methods, like the A* path planning algorithm, to determine the best paths while accounting for obstacles. It then uses cubic spline smoothing to guarantee seamless navigation. The ROS 2 system enables smooth communication between multiple nodes, thereby enabling the robot to operate independently in terms of navigation, material handling, and environmental interaction. The code's modular design, which includes path following, obstacle building, and map publishing, guarantees dependable operation and simple extension or

adaptation for future improvements. Overall, the code not only supports the robot's core functionalities but also enhances its performance, making it a robust solution for construction site automation.

5.Outlook

Various difficulties can exist when implementing the INNOK HEROS 223 robot in the construction industry. Even though expenses can be spread, its high initial cost in comparison to standard equipment may be a barrier for smaller businesses. Although teaching employees how to use the robot's hardware and software is essential, it can take some time and encounter opposition from staff members used to more traditional methods.

Another difficulty is the dynamic nature of construction sites, which might impact the robot's effectiveness because of things like uneven ground and barriers. To ensure efficient functioning, site modifications might be required, which would increase project complexity and cost. For businesses to be successful, they need to set up reliable communication systems for real-time monitoring, locate charging stations strategically, and guarantee a steady power supply [5].

Operators, technicians, and maintenance staff must receive comprehensive instruction, as well as regular updates on best practices and improvements. It is also essential to consider possible upgrades, such as hardware and software updates, to maximize the robot's performance and integrate it with other construction technologies.

By addressing these challenges through careful planning, infrastructure investment, and ongoing training, construction firms can leverage robotic technology to improve efficiency, productivity, and overall project outcomes.

6. References

[1] F. Barbosa, J. Woetzel, J. Mischke, M. J. Ribeirinho, M. Sridhar, M. Parsons, et al., "Reinventing construction: A route to higher productivity," McKinsey & Company, 2017. [Online]. Available: https://www.mckinsey.com/business-functions/operations/our-insights/reinventing-construction-through-a-productivity-revolution

[2] X. S. Dong, J. Wang, and C. Daw, "Fatal and nonfatal injuries among Hispanic construction workers, 1992-2008," CPWR Data Brief, vol. 2, no. 2, pp. 1-12, 2010. Available: https://stacks.cdc.gov/view/cdc/157529

[3] C. Hendrickson, "Labor, Material and Equipment Utilization," Project Management for Construction, Carnegie Mellon University, 2023. Available: https://www.cmu.edu/cee/projects/PMbook/04_Labor,_Material,_And_Equipment_Utilization.html.

[4] A. Research, "Design and Construction of Motorised Wheel Barrow," Afribary, 2018. Available: https://afribary.com/works/design-and-construction-of-motorised-wheel-barrow.

[5] Y. Liu, A. H. Alias, N. A. Haron, N. A. Bakar, and H. Wang, "Robotics in the Construction Sector: Trends, Advances, and Challenges," Journal of Intelligent & Robotic Systems, vol. 110, no. 72, p. PDF1, 2024. Available: https://link.springer.com/article/10.1007/s10846-024-02104-4

[6] Cubic Spline, https://mathworld.wolfram.com/CubicSpline.html#:~:text=A%20cubic%20spline%20is%20a,equations

[7] Bezier Curve , https://medium.com/data-science-shorts/what-is-a-bezier-curve-ba54935ef5b1

[8] *fusion2urdf-ros2 tool in our Project : https://github.com/dheena2k2/fusion2urdf-ros2*

Diagram01 Author's creation , System Architecture