Autonomous Exploration in 3D environments for Turtlebot

1st Loc Thanh Pham
Intelligent Field Robotics Systems
Universitat de Girona
Girona, Spain
u1992429@campus.udg.edu

2nd Atharva Anil Patwe
Intelligent Field Robotics Systems
Universitat de Girona
Girona, Spain
patweatharva@gmail.com

***Abstract*—This study aims to apply a method for autonomous exploration in intricate three-dimensional (3D) environments to a turtlebot platform using onboard sensors - Depth Camera and Lidar. In terms of global planner, we use Technologies for Autonomous Robot Exploration (TARE) planner while we apply Falco: Fast likelihood-based collision avoidance without a prior map for local planner. In experiments, our systems autonomously explore environments at a high degree of complexity, with simulation and turtlebot robot.**

***Index Terms*—TARE, Falco, Path Planning, Obstacle Avoidance, Exploration Strategies, Low-Level Controller, Stonefish, Turtlebot.**

I. Introduction

Autonomous exploration and path planning in unknown three-dimensional (3D) environments are critical challenges in the field of robotics. These tasks require robots to continuously update their understanding of the environment and plan traversable paths through complex spaces while managing computational constraints effectively. In this report, we present our implementation of an integrated approach that combines two state-of-the-art algorithms on a TurtleBot robot to enhance its autonomous exploration and path planning capabilities. This integration leverages the strengths of both algorithms to create a more robust and efficient system for navigating unknown environments.

TARE planner utilizes a hierarchical framework to improve exploration efficiency. It consists of two levels: one maintains dense data for detailed path planning within a local scope, and the other maintains sparse data for broader, global path planning. This approach leverages the insight that detailed processing is most effective near the robot and achieves computational efficiency by reducing the focus on distant details. The method optimizes the overall exploration path concerning its length and ensures the local path is kinodynamically feasible for the vehicle to navigate at high speeds.

The local planner Falco addresses the path planning problem with a focus on enabling fast autonomous navigation in complex environments. Originally designed for lightweight aerial vehicles, this method has been adapted for the TurtleBot. This local planner tracks the path coming from global planner and avoids obstacles. This approach reduces computational complexity by formulating the planning problem from a likelihood perspective, aiming to maximize the probability of successful navigation rather than minimizing path length. This probabilistic strategy allows the robot to prefer open spaces, offering greater flexibility in dynamic environments.

By integrating these two algorithms, we leverage the strengths of both approaches to achieve a more robust and efficient exploration and navigation system for the TurtleBot. The integrated system combines the detailed local path planning and global path optimization from the TARE with the probabilistic, low-complexity path planning of the Falco algorithm.

We evaluate the performance of our integrated system in both simulated and real-world environments. Our experiments demonstrate significant improvements in exploration efficiency and path planning performance. The TurtleBot, equipped with the integrated planner, achieves a higher explored volume per second and navigates complex environments with reduced computational demands.

This report details our methodology, implementation, and evaluation of the integrated algorithms on the TurtleBot. Additionally, we provide a software environment for benchmarking exploration algorithms, which includes simulation models, navigation modules, and visualization tools. Our algorithm code is publicly available to support further research and development in autonomous navigation systems.

II. Methodology

A. Global Planner: TARE

1) Local Planner: In Figure 2, there are 6 steps to compute local path. First, upon receiving sensor scans, the algorithm derives collisions from these scans and simultaneously downsamples the data to reduce computational load. Collision derivation is based on the height and slope of objects. Second, from the sensor scans and derived collisions, the algorithm identifies free spaces and occupied spaces. Subsequently, the frontier is calculated and identified based on these spaces. Specifically, the frontier is the boundary between unknown space and free space. Third, the algorithm samples viewpoints within the local planning horizon. Viewpoints that satisfy the conditions of no collision, connectivity, and being within free space are selected as candidate viewpoints. Fourth, based on the set of candidate viewpoints and the identified frontier, the algorithm

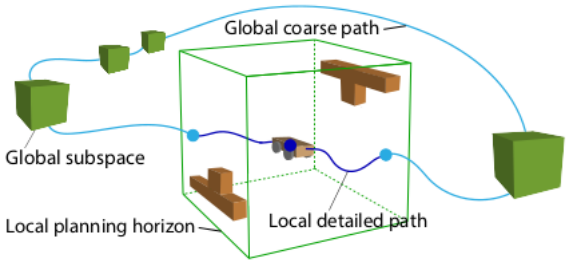

Fig. 1. Illustration of our exploration framework. Inside the local planning horizon (green box), data is densely maintained and a local detailed path is computed (dark-blue curve). At the global scale, data is sparsely maintained in distant subspaces (solid green cubes) and a global coarse path is computed (light-blue curve). The local path and global path are connected on the boundary of the local planning horizon to form the exploration path.

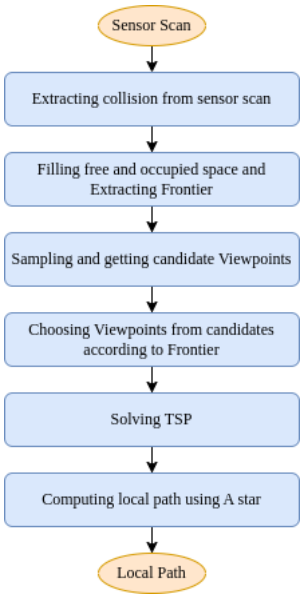

Fig. 2. Local Planner flow chart.

calculates the number of frontier points each candidate viewpoint can observe, using sensor parameters such as distance and elevation angle. The candidate viewpoints are then sorted in descending order based on the number of frontier points they can observe, and viewpoints are selected starting from the highest rank. Note that after selecting a viewpoint, the frontiers observed by that viewpoint are excluded from the remaining frontiers to be observed by subsequent viewpoints. The selection process concludes when all frontiers have been observed or all candidate viewpoints have been selected. Fifth, the Traveling Salesman Problem (TSP) is applied to the list of selected viewpoints to determine the optimal path that passes through all viewpoints, thus determining the sequence of viewpoints along the path. Finally, to calculate the path between two viewpoints, the A* algorithm is used to find the optimal path. The A* algorithm is applied to a graph of the selected candidate viewpoints, ensuring connectivity and precomputed distances between viewpoints. This results in a complete local path.

2) Global Planner: The exploration space outside the local planning horizon is divided into equal cuboid subspaces. Each subspace stores data about covered and uncovered surfaces. This data is kept for storage purposes, while the data within is actively updated during exploration. Each subspace has one of three statuses: "unexplored", "exploring", or "explored". A subspace is "unexplored" if it contains no surfaces, "explored" if it only contains covered surfaces, and "exploring" if it contains any uncovered surfaces. Only the "exploring" subspaces are considered in global planning.

The global planning objective is to find a global path that traverses the current viewpoint and the centroid of each exploring subspace. A sparse random roadmap is constructed in the traversable space expanded from the past trajectory, and the A* algorithm is used to find the shortest paths between subspaces. The overall procedure involves constructing a distance matrix of paths on the roadmap, solving a TSP to order the subspaces, and then connecting these paths to form the global path.

First, it constructs a distance matrix and solves a TSP to determine the order of subspaces, in Figure 4. Two points where global path intersects the boundary of local planning horizon are identified. Local planner is then used to compute the local path. Finally, local path and global path are concatenated to become exploration path.

When the local path within is complete, the vehicle transitions to these subspaces to continue exploration. The algorithm implicitly manages the transition between exploration and relocation.

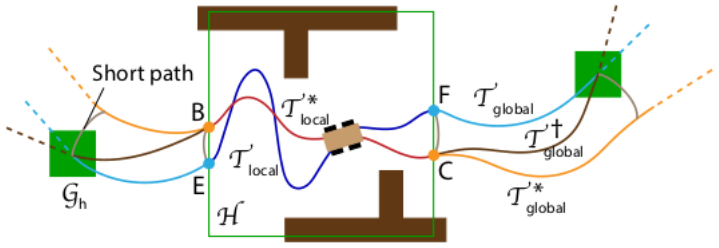

Fig. 3. Global Planner

B. Local Planner: Falco

The method aims to maximize the likelihood of a vehicle successfully navigating from point A to point B. Obstacles within the sensor range are considered deterministically known based on perception sensors, while obstacles beyond this range are treated probabilistically if a prior map is available. This dual representation allows the method to efficiently handle environmental uncertainties. The probability density function for the vehicle to reach B from an initial state x_s is denoted as $p(x_B \mid x_s)$.

Given the start state x_s, a path is defined as the trajectory connecting points A and B. All possible paths must intersect with the sensor frontier, represented by $\mathcal{F}$. The state of the

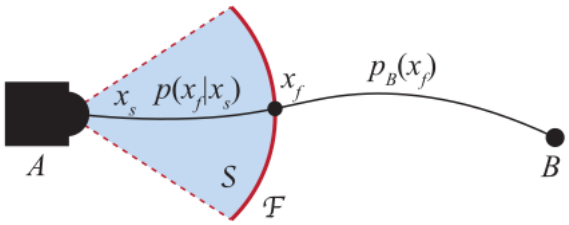

Fig. 4. Illustration of sensor range as the gray area and sensor frontier as the red solid curve. The black curve is a path to navigate from A to B , which starts at x_s and intersects with $\mathcal{F}$ at x_f

vehicle while passing $\mathcal{F}$ is denoted as x_f. The conditional distribution of x_f given x_s, $p(x_f \mid x_s)$, is derived from the obstacle information provided by the perception sensors. The probability density for the vehicle to reach B from x_f, $p(x_B \mid x_f)$, can be calculated based on the obstacles on the prior map.

To approximate the probability density, Monte Carlo sampling is used. For n samples drawn from $p(x_f \mid x_s)$, the probability density $p(x_B \mid x_s)$ can be approximated as:

$$p(x_B \mid x_s) \approx \frac{1}{n} \sum_{i=1}^{n} p(x_B \mid \xi_i) \tag{1}$$

where ξ_i are the samples of x_f.

1) Local Probabilities: The method utilizes path groups, which are sets of paths sharing the same start state x_s. Each path in a group is generated as a cubic spline curve, splitting into multiple directions both horizontally and vertically. For instance, if there are 7 horizontal and 5 vertical directions, a total of 35 path groups can be created, resulting in 42,875 paths. During navigation, obstacles detected by the perception sensors occlude certain paths, and the path clearance is indicated by a Boolean function $c(\xi_i)$.

The probability $P(x_B \mid x_s)$ is computed as:

$$P(x_B \mid x_s) \approx \frac{\sum_{i=1}^{n} c(\xi_i) p(x_B \mid \xi_i)}{\sum_{i=1}^{n} c(\xi_i)} \tag{2}$$

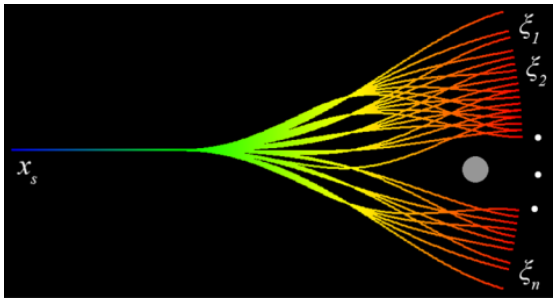

Fig. 5. Collision-free paths in a group with an obstacle as the gray dot. The paths start at x_s

2) Global Probabilities: The environment is represented using voxels that include both position and orientation information. Each voxel state $x_{j,k}$ includes a voxel index j and a direction index k. The probability density to reach B from $x_{j,k}$, $p(x_B \mid x_{j,k})$, is calculated iteratively. During initialization, probability densities are evenly distributed among all directions within the voxel containing B. The algorithm iterates through the paths to update the probability densities until the changes fall below a threshold.

III. Implementation and Results

While implementing the mentioned methodology, two ROS nodes were developed. - Global Planner, Local Planner. The overall architecture is depicted in the figure 6.

A. Lidar Plugin

Initially, we intended to use a depth camera for the system to obtain a 3D point cloud. However, during development, we encountered problems synchronizing depth camera data and estimated state of the robot, so we decided to use lidar as a sensor for the system. Since lidar provides 2D data while the system evolves with 3D, we further developed this plugin, which helps convert lidar's 2D data into 3D data, in Figure 7. Basically, the plugin extends the obtained 2D point cloud in the Z direction to get elevation data. This is a situational solution, suitable for lab testing environments. Currently, we are still looking for a solution to synchronize depth camera data and robot status.

B. Global Planner

Global planner is calculated every 1 second and sends next way point to local planner. Output of the local planner is presented in Figure 8 while Figure 9 is output of the glocbal planner.

C. Local Planner

The path groups are pre-generated offline, and a voxel grid is used for collision checks. The correspondences between voxels and paths are pre-established and stored in an adjacency list. During operation, perception sensor data points are processed to label occluded paths. The algorithm then selects the path group with the highest probability $P(x_B \mid x_s)$. The computational complexity of the online processing algorithm is $O(mnh)$, where m is the number of perception sensor data points, n is the number of paths in a path group, and h is the number of path groups. Result of the local planner is showed in Figure 10.

IV. Conclusion

In this study, we successfully implemented an autonomous exploration system on a Turtlebot platform using ROS. Our approach integrated TARE planner for global planner and Falco planner for local planner. While global planner processed 3D point cloud from sensor scan and robot's states estimated from localization system to generate path, local planner created a detailed path which satisfies the kinodynamic constraints of the vehicle to tracks that path. This system showed highly efficient

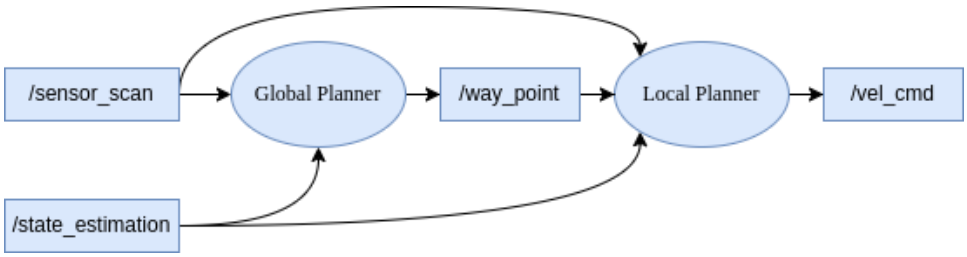

Fig. 6. Planning ROS Package Architecture

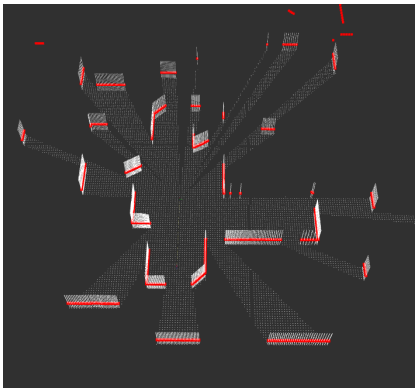

Fig. 7. The red line is the Lidar measurement while white point cloud is output of the Lidar plugin.

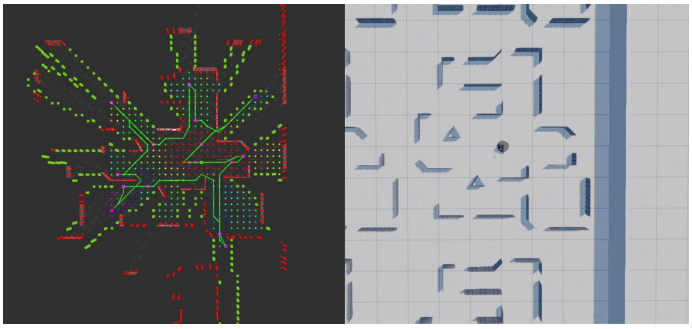

Fig. 8. The white point cloud is sensor scan; The red point cloud is collision extracted from sensor scan; The green point cloud is frontier; The colored point cloud on the ground is candidate viewpoints, viewpoints which are brighter color are considered to see more frontier; The green cubes are selected viewpoints; The green line is local path.

exploration of large and complex environments. We tested this system in both StoneFish simulation and real turtlebot robot in laboratory environment.

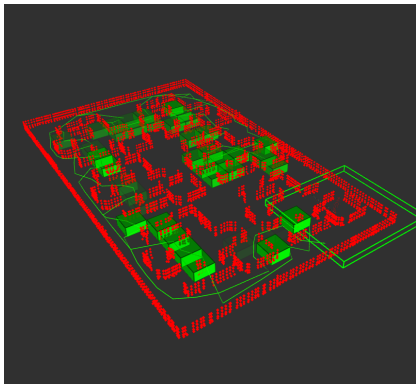

Fig. 9. Green cubes are subspace in the global planner; The green line connecting all subspace and local planning horizon is the global path

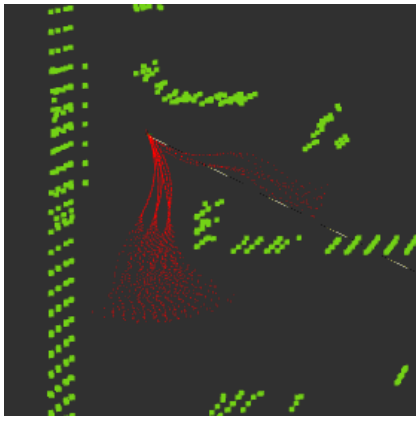

Fig. 10. The green point cloud is collision; The red point cloud on the ground is feasible path from local planner.

APPENDIX

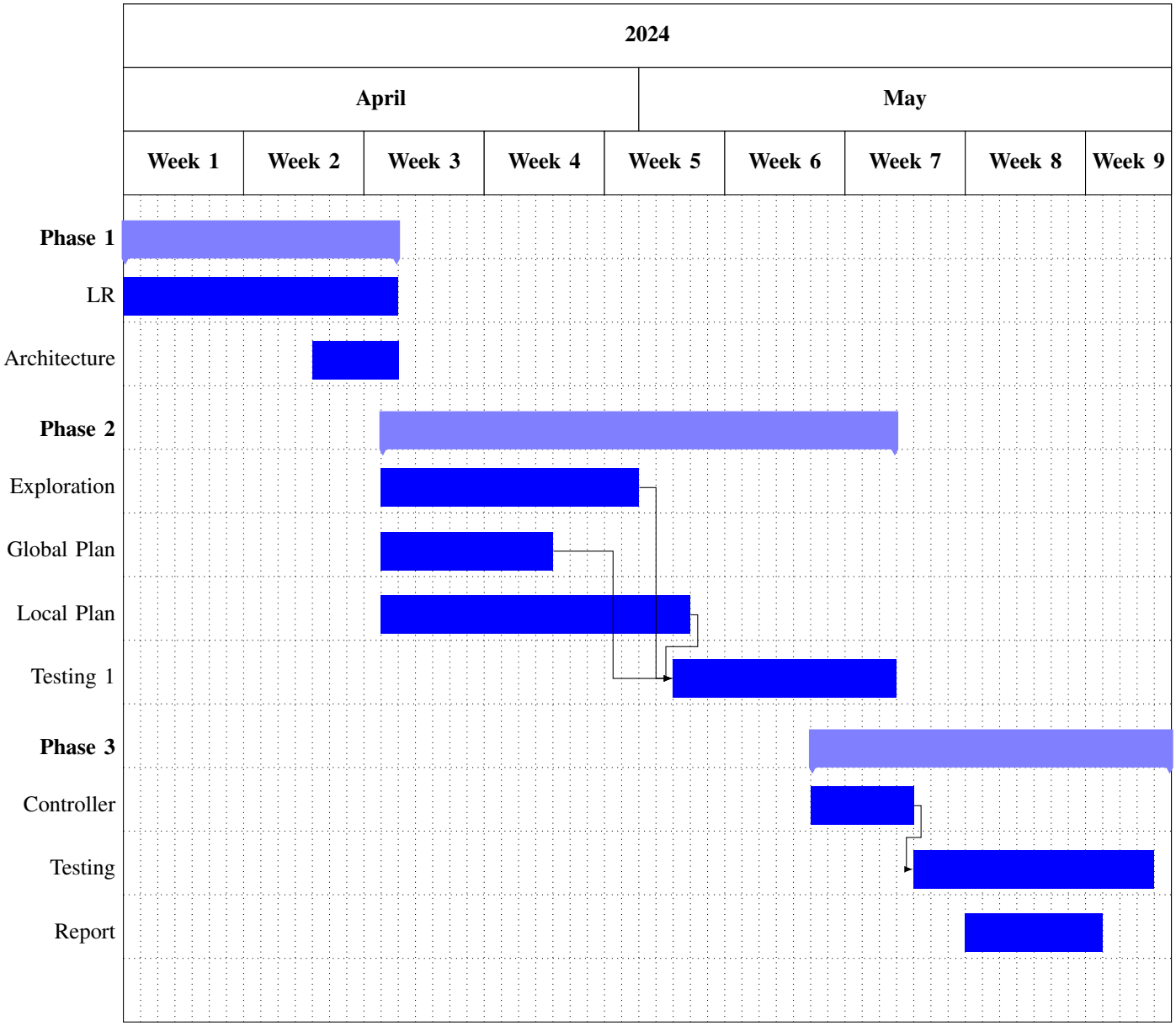

Fig. 11. Gantt Chart For the Hands On Project Planned

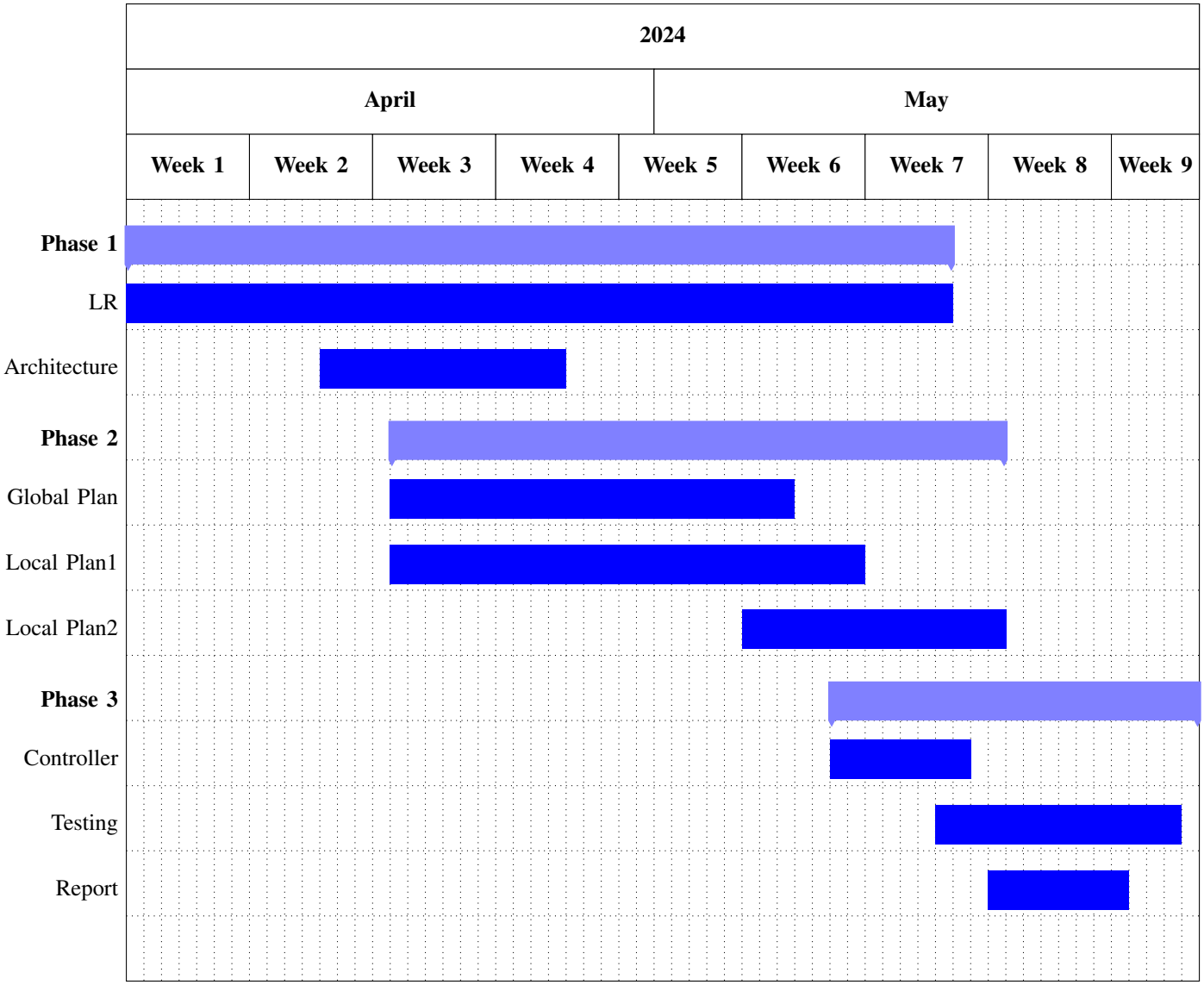

Fig. 12. Gantt Chart For the Hands On Project Actual (Deviations)