HW1: State Estimation in 2D using ROS2

RAS 598: Space Robotics and AI

Aldrin Inbaraj A
ASU ID: 1226200393

Problem Description

The objective of this assignment is to perform 2D state estimation using ROS2 and to evaluate the performance of a PD controller in a simulated environment. The assignment involves tuning the proportional and derivative gains to minimize the cross-track error while following a boustrophedon path. Additionally, the effect of the spacing parameter on coverage is analyzed.

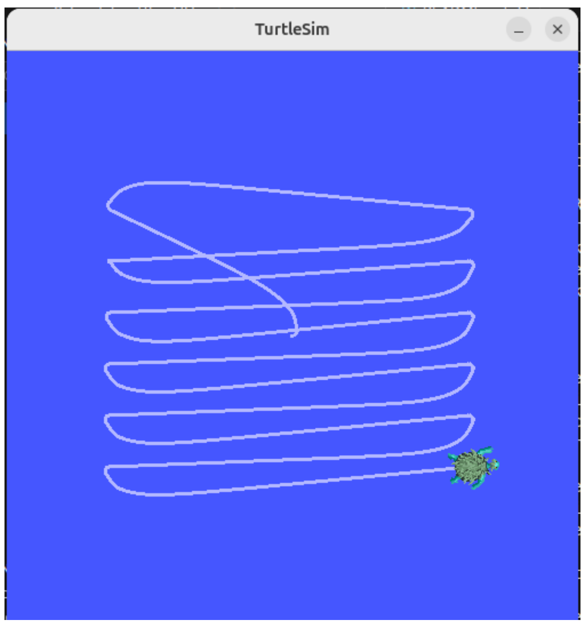

Figure 1: Path that has to be optimized by tweaking PD gains and spacing

The above image corresponds to the following parameters:

- `Kp_linear`: 10.0
- `Kd_linear`: 0.1
- `Kp_angular`: 5.0
- `Kd_angular`: 0.2
- `spacing`: 1.0

Methodology

Real-Time Tuning with RQT

- Used `rqt_plot` to visualize performance metrics such as cross-track error.
- Utilized `rqt_configure` to modify the PD gains and observe their effect in real-time.
- Observations:
  - Tweaking `Kp_linear`:
    - Increasing `Kp_linear` results in faster convergence to the desired trajectory but can lead to overshooting and oscillations if too high.
    - Decreasing `Kp_linear` results in slower convergence and a less responsive system.
  - Tweaking `Kd_linear`:
    - Increasing `Kd_linear` helps dampen oscillations and stabilize the response but can slow down trajectory corrections.
    - Decreasing `Kd_linear` reduces stability and can result in oscillatory behavior.
  - Tweaking `Kp_angular`:
    - Increasing `Kp_angular` improves the system's ability to quickly correct angular errors but may cause sharp, abrupt movements if too high.
    - Decreasing `Kp_angular` leads to slower angular error corrections and a less responsive trajectory.
  - Tweaking `Kd_angular`:
    - Increasing `Kd_angular` stabilizes the angular response but may slow down corrections.
    - Decreasing `Kd_angular` can lead to jerky angular adjustments and instability.
  - Adjusting `spacing`:
    - Smaller spacing results in better path coverage but increases the computational load and requires tighter control to avoid overshooting.
    - Larger spacing reduces computational load and trajectory adjustments but leads to gaps in coverage.
- Example Case:
  - With the following parameters:
    - `Kp_linear = 2.0`, `Kd_linear = 0.1`
    - `Kp_angular = 3.0`, `Kd_angular = 0.8`
    - `spacing = 2.0`
  - The trajectory exhibited overcorrection, where the system oscillated excessively while attempting to correct its path.

- Plotted and observed the system's performance in the following figures:

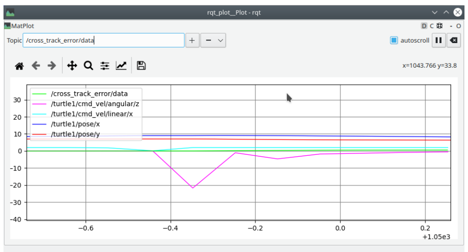

Figure 2: RQT Plot of Cross-Track Error

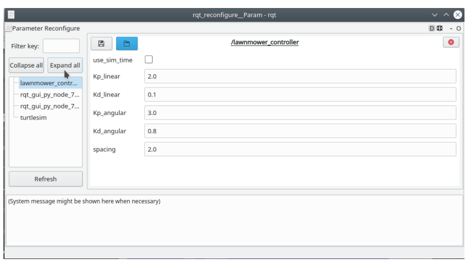

Figure 3: RQT Reconfigure Tool for Tuning Gains

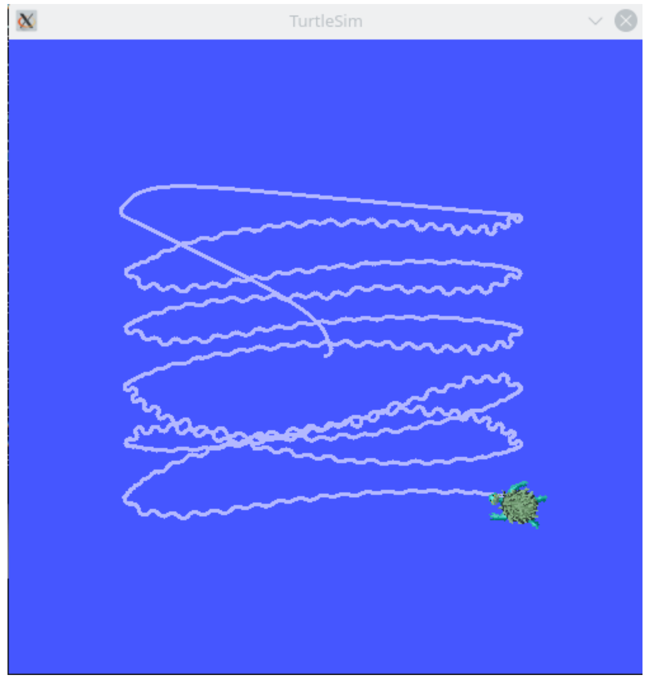

Figure 4: Resulting Trajectory in Turtlesim

Bayesian Optimization

Bayesian Optimization is a powerful tool for optimizing functions that are expensive to evaluate. It uses a probabilistic model, typically a Gaussian Process, to estimate the objective function and iteratively selects parameters that are likely to minimize (or maximize) the objective.

- Why it is useful here:
  - Optimizing PD gains manually can be time-consuming and non-intuitive, especially when multiple parameters interact in complex ways.
  - Bayesian Optimization efficiently searches the parameter space to find a near-optimal solution with fewer evaluations compared to grid or random search.
- Application in this assignment:
  - Used Bayesian Optimization to minimize the cross-track error by systematically tuning the `Kp` and `Kd` parameters for both linear and angular controllers.
  - The optimizer focuses on high-value regions in the parameter space, ensuring better performance while reducing manual effort.

Implementation

Optimizer Code

- A new file, `boustrophedon_optimizer.py`, was created to optimize the PD gains for a specific spacing value.
- The optimization minimizes cross-track error using Bayesian Optimization from the `optuna` library. The process involves fixing the `spacing` parameter and tuning the PD gains (`Kp_linear`, `Kd_linear`, `Kp_angular`, and `Kd_angular`) for a given trajectory.
- To determine the optimal spacing, multiple optimizations are performed for different `spacing` values. This ensures the best balance between coverage and minimal cross-track error while ensuring the PD gains are appropriate for each spacing.
- The best parameters for each optimization run are saved in a text file for future reference. The following code snippet explains the key components of the optimization process:

Python Code

```python
import optuna

class BoustrophedonOptimizer(Node):
    def optimize_gains(self):
        def objective(trial):
            # Suggest PD gains for optimization
            Kp_linear = trial.suggest_float('Kp_linear', *self.KP_LINEAR_RANGE)
            Kd_linear = trial.suggest_float('Kd_linear', *self.KD_LINEAR_RANGE)
            Kp_angular = trial.suggest_float('Kp_angular', *self.KP_ANGULAR_RANGE)
            Kd_angular = trial.suggest_float('Kd_angular', *self.KD_ANGULAR_RANGE)

            # Simulate controller and compute average cross-track error
            avg_error = self.simulate_controller(Kp_linear, Kd_linear, Kp_angular, Kd_angular)
            return avg_error

        # Fix the spacing for this optimization
        self.spacing = 0.5  # Example spacing
        self.get_logger().info(f"Running optimization for spacing={self.spacing}")

        # Run Bayesian Optimization with optuna
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=self.n_trials)

        # Save the best parameters
        best_params = study.best_params
        best_value = study.best_value
        self.get_logger().info(f"Best parameters: {best_params}")
        self.get_logger().info(f"Best average cross-track error: {best_value:.3f}")

        # Save results to a text file
        with open('best_params.txt', 'w') as f:
            f.write(f"Best parameters: {best_params}\n")
            f.write(f"Best average cross-track error: {best_value:.3f}\n")

        return best_params
```

Listing 1: Optimizer Code Snippet

- The `objective` function defines the optimization target, minimizing the average cross-track error by systematically tuning the PD gains.

- The `spacing` parameter is kept constant during each optimization run, as seen in `self.spacing = 0.5`. To find the best spacing value, multiple optimizations are performed by updating `self.spacing`.
- Once the optimization is complete:
  - The best PD gains and their corresponding cross-track error are logged.
  - The best parameters are saved in a text file (`best_params.txt`) for future use.

Workflow

1. Run the optimizer to determine the best parameters using Bayesian Optimization.
2. Save the best parameters and their corresponding cross-track error in a text file for future reference.
3. Output from the `boustrophedon_optimizer` is shown below:
   - Output:

```
[INFO] [1737972824.622087790] [boustrophedon_optimizer]:
Best parameters found: {'Kp_linear': 5.0014964543025275,
'Kd_linear': 0.3440134687670172,
'Kp_angular': 9.299200927443698,
'Kd_angular': 0.10005046439984644}
[INFO] [1737972824.622312004] [boustrophedon_optimizer]:
Best average cross-track error: 0.205

```

Listing 2: Optimizer Output

4. The best parameters are then applied to the controller, and the final average cross-track error is calculated.

Results

- Output from `boustrophedon_controller.py`:

```
[boustrophedon_controller-2] [INFO] [1738046772.783766592] [lawnmower_controller
    ]:
Cross-track error - Current: -0.003, Avg: 0.088, Min: 0.000, Max: 0.195, Segment
    : (2.0,3.0)->(9.0,3.0)
[boustrophedon_controller-2] [INFO] [1738046772.785660082] [lawnmower_controller
    ]:
Reached waypoint 22
[boustrophedon_controller-2] [INFO] [1738046772.883342999] [lawnmower_controller
    ]:
Lawnmower pattern complete
[boustrophedon_controller-2] [INFO] [1738046772.885677395] [lawnmower_controller
    ]:
Final average cross-track error: 0.088

```

Listing 3: Controller Output

- Final Gains and Spacing selected which yields a final average cross track error of 0.088.
  - `Kp_linear: 5.0015`
  - `Kd_linear: 0.3440`
  - `Kp_angular: 9.2992`
  - `Kd_angular: 0.1000`
  - `Spacing: 0.5`
- Image of the resulting trajectory:

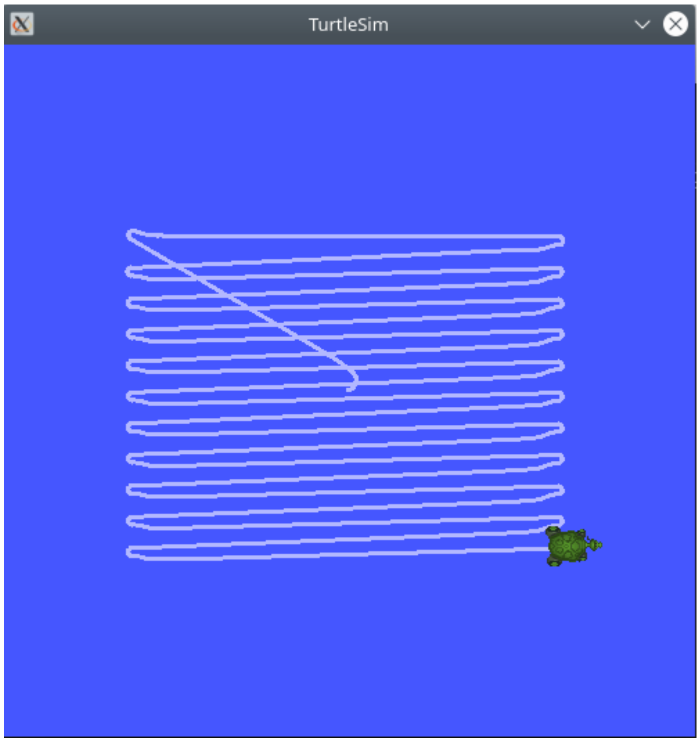

Figure 5: Resulting Trajectory with Optimized PD Gains and Spacing

- Comparison Between Optimizer and Controller Outputs:
  - The optimizer suggested a set of parameters that minimized the simulated average cross-track error to approximately `0.205`.
  - When these parameters were applied in the actual controller, the final average cross-track error was measured as `0.088`.
  - The improvement in the controller's performance compared to the optimizer's simulation may be attributed to real-time adjustments and better handling of dynamic factors by the controller.

- Reasoning for Spacing Selection:
  - Comparison of different spacing values:

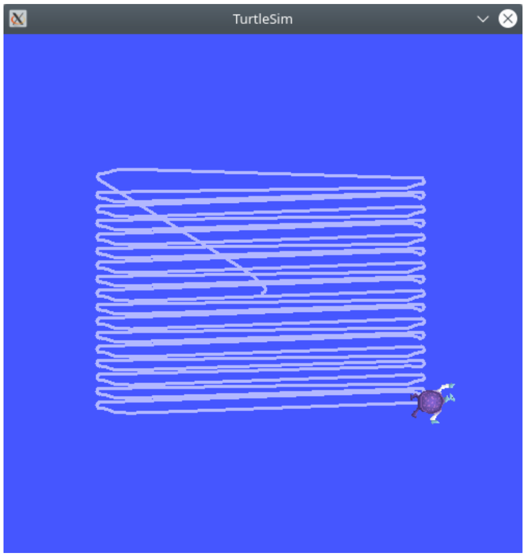

Figure 6: Spacing = 0.3

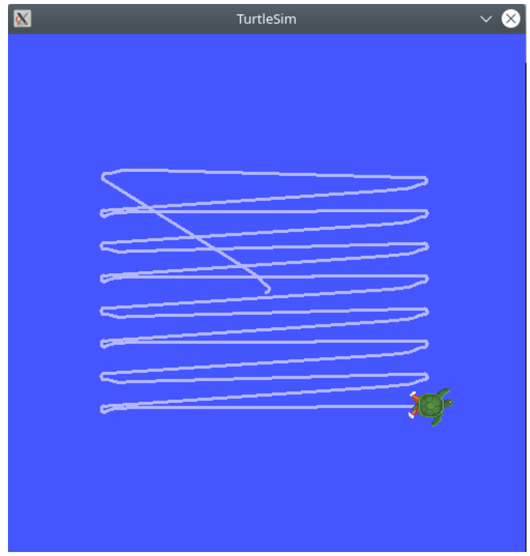

Figure 7: Spacing = 0.7

- The spacing value was kept constant during each optimization. However, for different spacing values, the PD controllers had to be re-optimized.
- After testing multiple spacing values and optimizing the gains for each, a spacing value of `0.5` was selected.
- At `spacing = 0.5`, the paths were:
  - Not too close, avoiding overlap and unnecessary corrections.
  - Not too far apart, ensuring adequate coverage without gaps in the lawnmower pattern.

Performance Plots

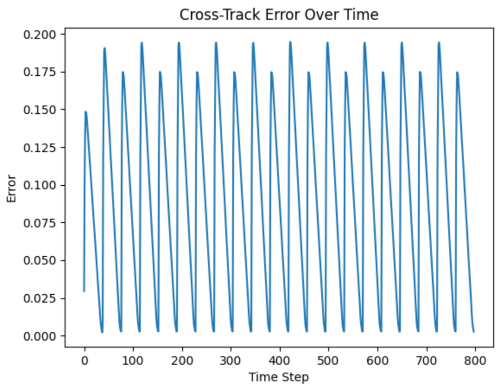

Figure 8: Cross-Track Error Over Time

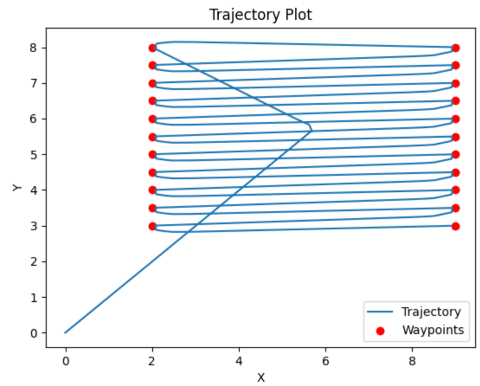

Figure 9: Trajectory Plot

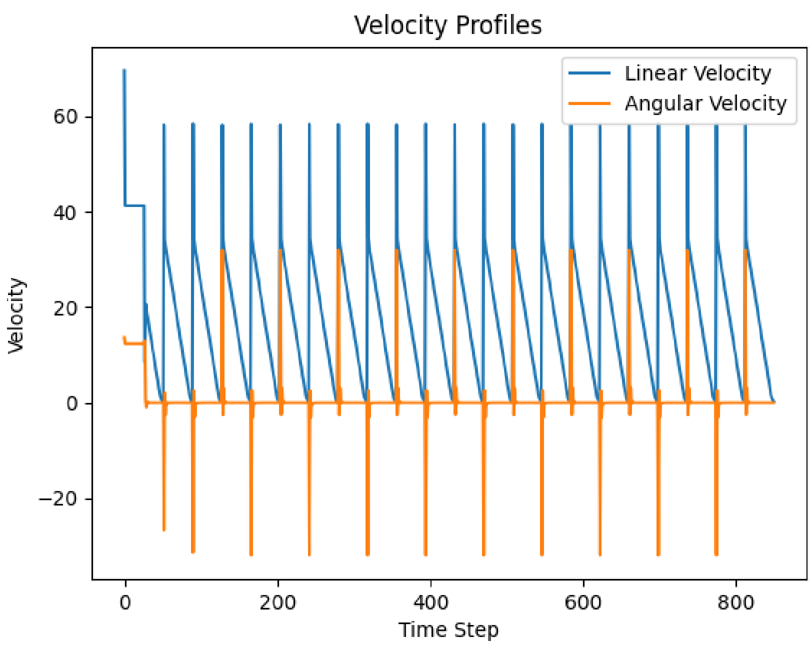

Figure 10: Velocity Profiles (Linear and Angular)

Challenges and Observations

- **Difficulties in Tuning Gains Manually:**
  - Small changes in `Kp` values led to large variations, making manual tuning difficult.
- **Impact of Spacing on Coverage:**
  - Small spacing (0.3) caused overlapping paths, while large spacing (0.7) left coverage gaps.
  - Spacing = 0.5 was optimal, balancing coverage without redundancy.
  - PD gains needed re-optimization for each spacing value.
- **Effectiveness of Bayesian Optimization:**
  - Automated tuning found optimal gains, reducing manual effort.
  - Optimized simulation error: 0.205; real controller error improved to 0.088.
  - Best gains were saved for reuse, improving efficiency.
- **Stability vs. Responsiveness Trade-off:**
  - High `Kp` improved response but increased oscillations.
  - Higher `Kd` reduced overshooting but slowed corrections.
  - Final selected gains balanced precision and stability.
- **Insights from Plots:**
  - Cross-Track Error Plot showed periodic oscillations.
  - Trajectory Plot revealed improved path following.
  - Velocity Profiles indicated rapid corrections due to PD control.