EE4308 – PROJECT 1

Leonardo Groot
Zaw Hein Aung
Hsan Ennouri
Nie Zhen

INTRO

1. Planner
2. Controller
3. Parameter Tuning
4. Lab Results

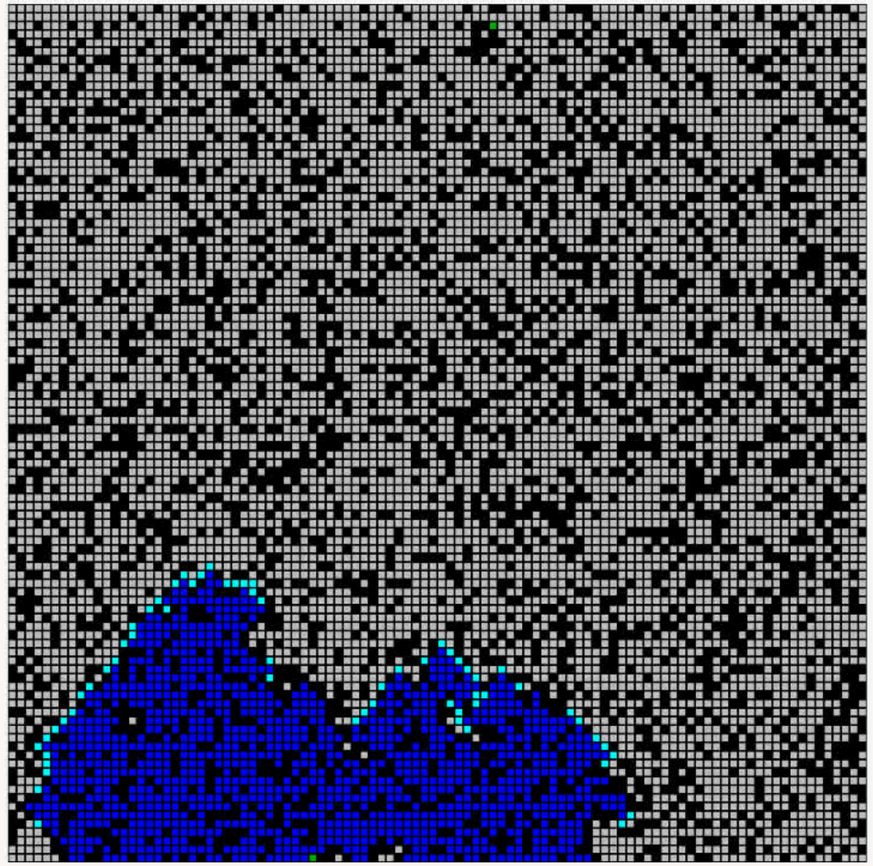
PATH PLANNING

ALGORITHMS

- A*
- A* with Savtsky-Golay smoothing
- A* with post-processing
- Theta*

A*

- Optimal and complete
- Efficient
- Flexible



A*

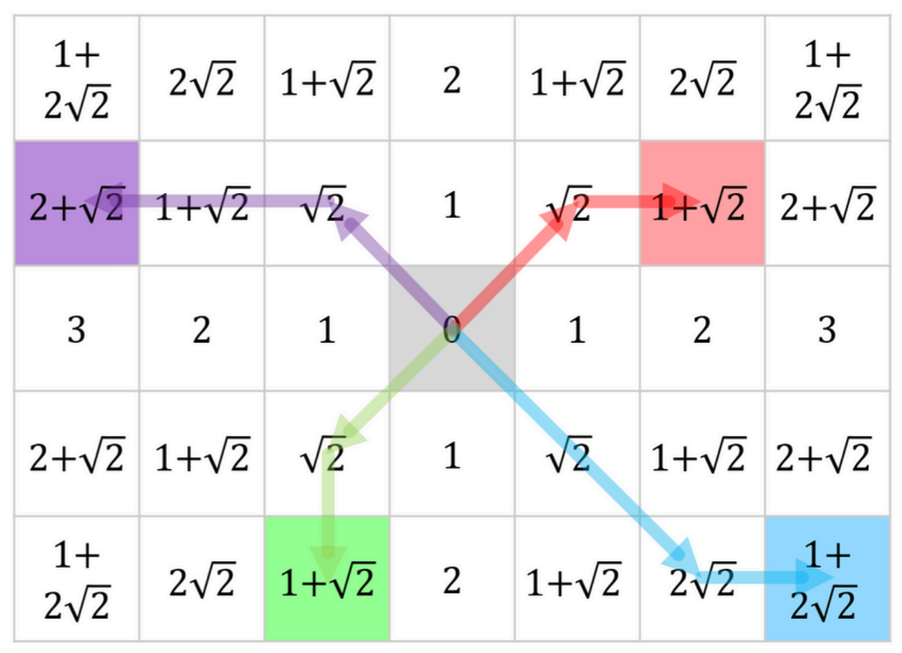
g -> cost of movement

h -> distance to goal

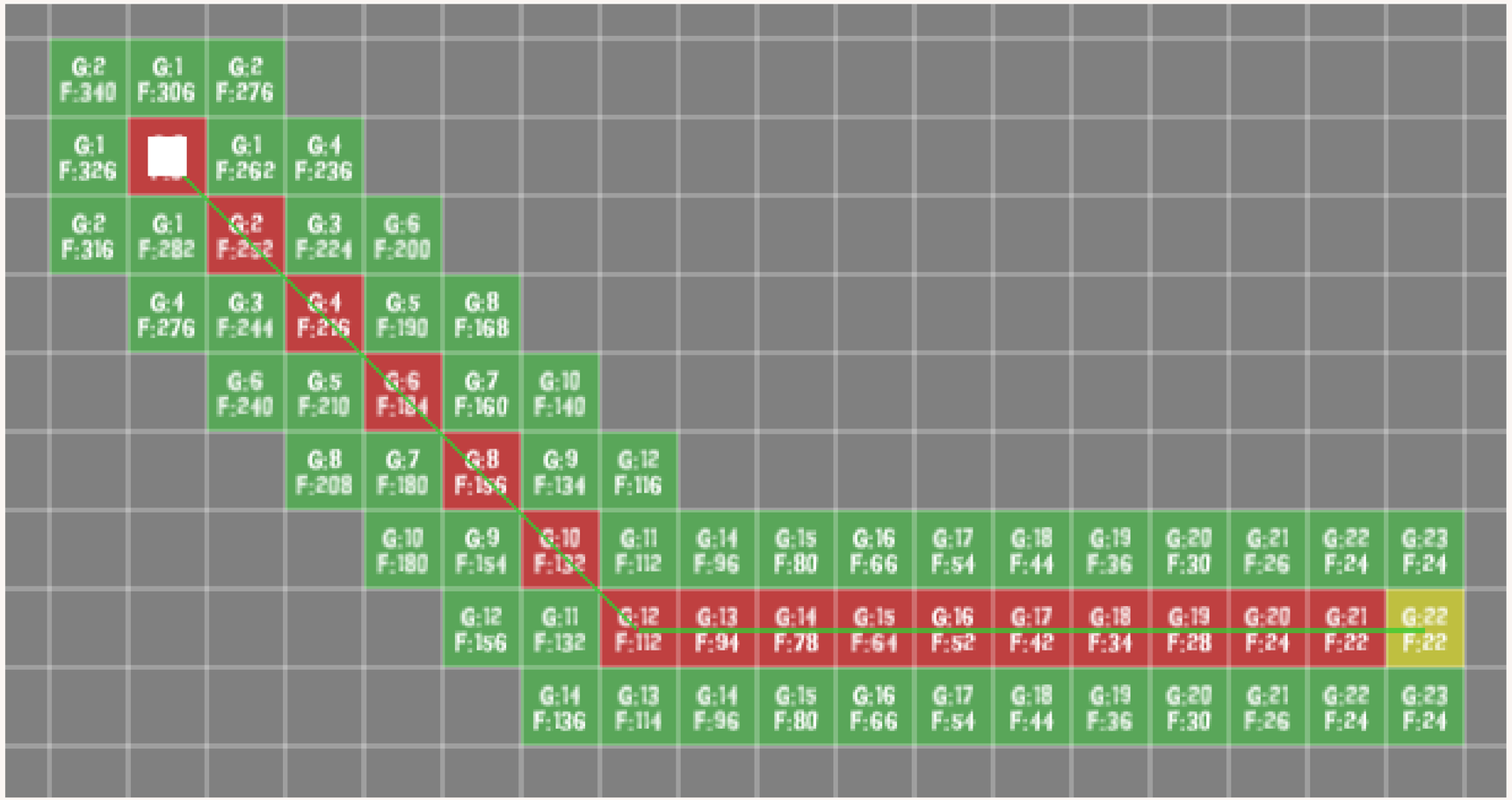
f -> *g* + *h*

Euclidean distance

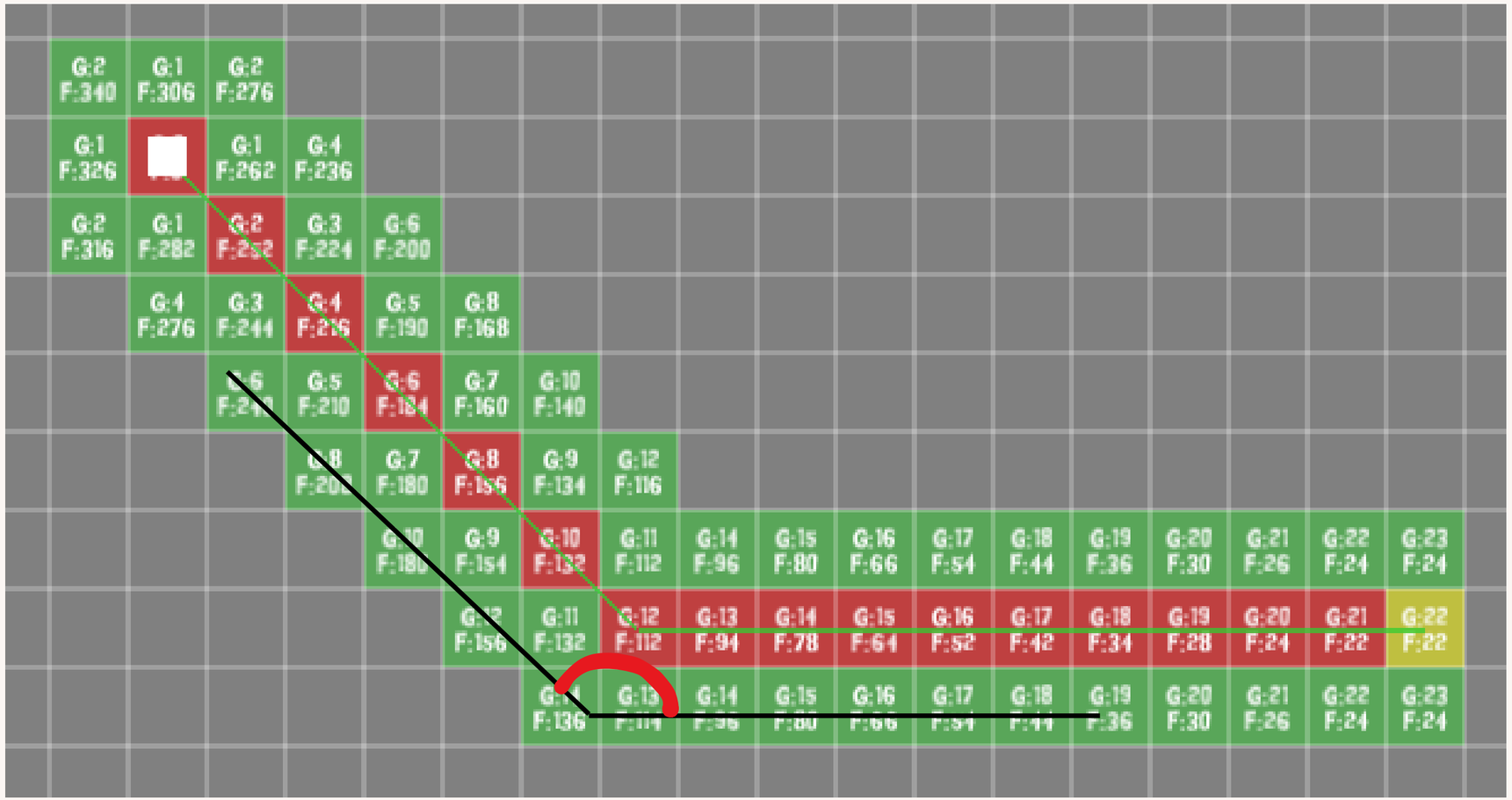
8-connected grid



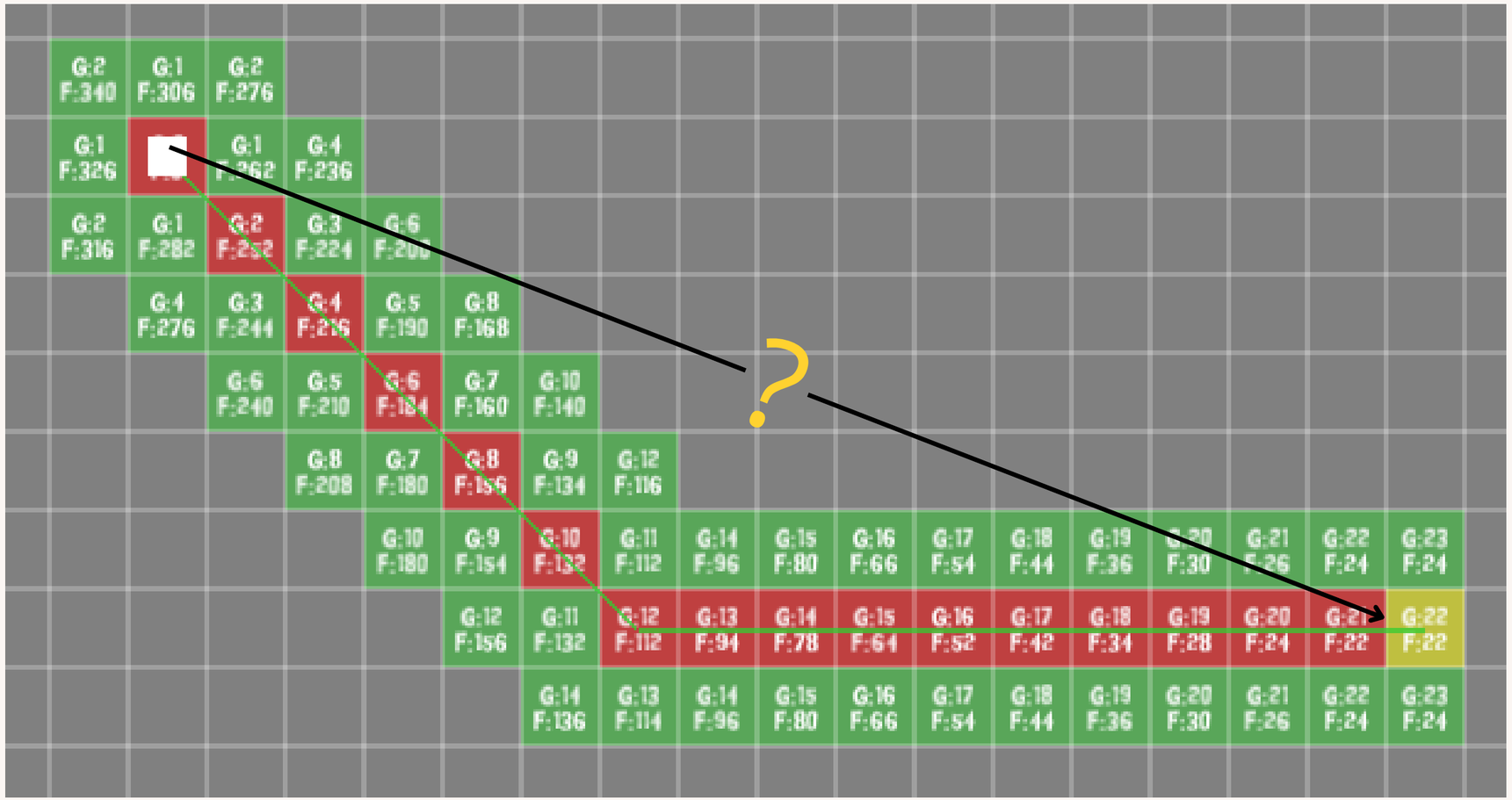
A*


G:2 F:340
G:1 F:306
G:2 F:276
G:1 F:326
G:1 F:262
G:4 F:236
G:2 F:316
G:1 F:282
G:2 F:252
G:3 F:224
G:6 F:200
G:4 F:276
G:3 F:244
G:4 F:216
G:5 F:190
G:8 F:168
G:6 F:240
G:5 F:210
G:6 F:184
G:7 F:160
G:10 F:140
G:8 F:208
G:7 F:180
G:8 F:156
G:9 F:134
G:12 F:116
G:10 F:180
G:9 F:154
G:10 F:132
G:11 F:112
G:14 F:96
G:15 F:80
G:16 F:66
G:17 F:54
G:18 F:44
G:19 F:36
G:20 F:30
G:21 F:26
G:22 F:24
G:23 F:24
G:12 F:156
G:11 F:132
G:12 F:112
G:13 F:94
G:14 F:78
G:15 F:64
G:16 F:52
G:17 F:42
G:18 F:34
G:19 F:28
G:20 F:24
G:21 F:22
G:22 F:22
G:14 F:136
G:13 F:114
G:14 F:96
G:15 F:80
G:16 F:66
G:17 F:54
G:18 F:44
G:19 F:36
G:20 F:30
G:21 F:26
G:22 F:24
G:23 F:24

A*

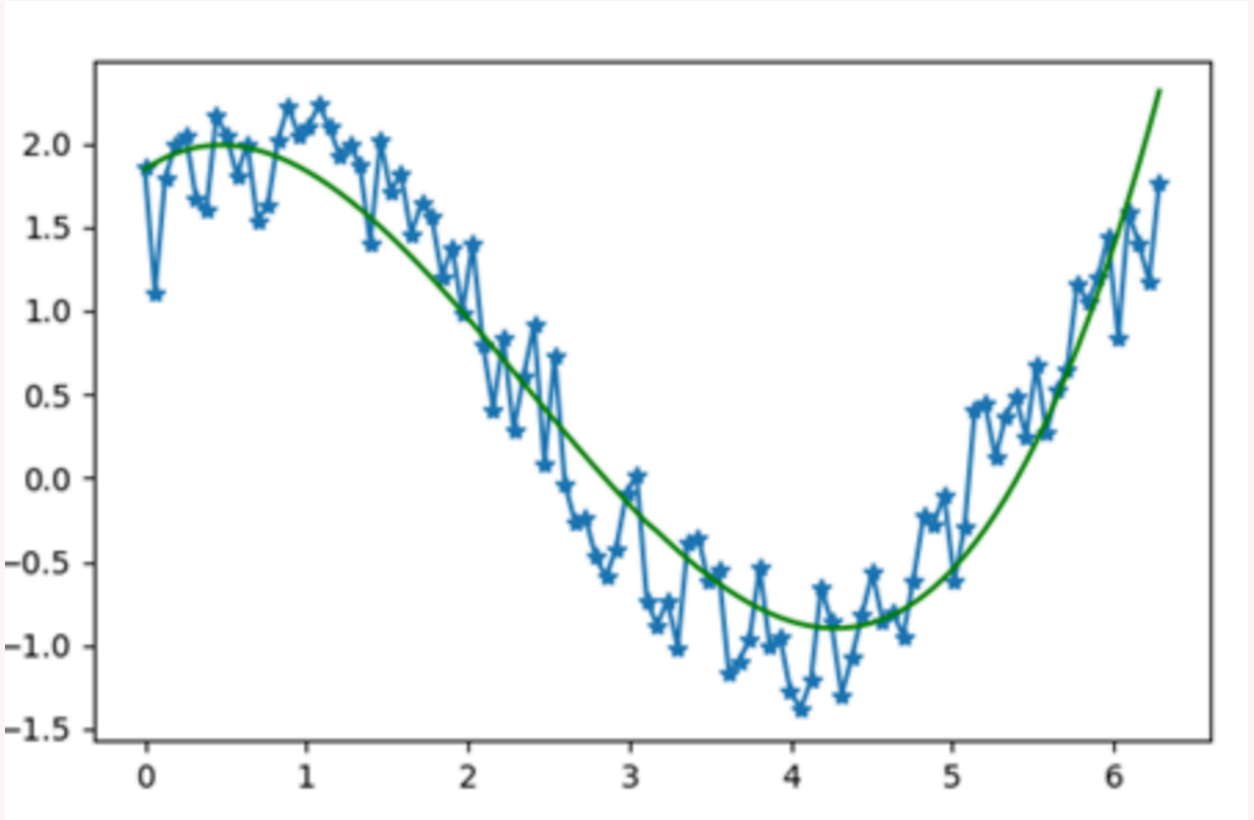

G:2 F:340
G:1 F:306
G:2 F:276
G:1 F:326
G:1 F:262
G:4 F:236
G:2 F:316
G:1 F:282
G:3 F:224
G:6 F:200
G:4 F:276
G:3 F:244
G:5 F:190
G:8 F:168
G:5 F:210
G:7 F:160
G:10 F:140
G:7 F:180
G:9 F:134
G:12 F:116
G:9 F:154
G:11 F:112
G:14 F:96
G:15 F:80
G:16 F:66
G:17 F:54
G:18 F:44
G:19 F:36
G:20 F:30
G:21 F:26
G:22 F:24
G:23 F:24
G:11 F:132
G:12 F:112
G:13 F:94
G:14 F:78
G:15 F:64
G:16 F:52
G:17 F:42
G:18 F:34
G:19 F:28
G:20 F:24
G:21 F:22
G:22 F:22
G:14 F:136
G:15 F:80
G:16 F:66
G:17 F:54
G:18 F:44
G:19 F:36
G:20 F:30
G:21 F:26
G:22 F:24
G:23 F:24

A*

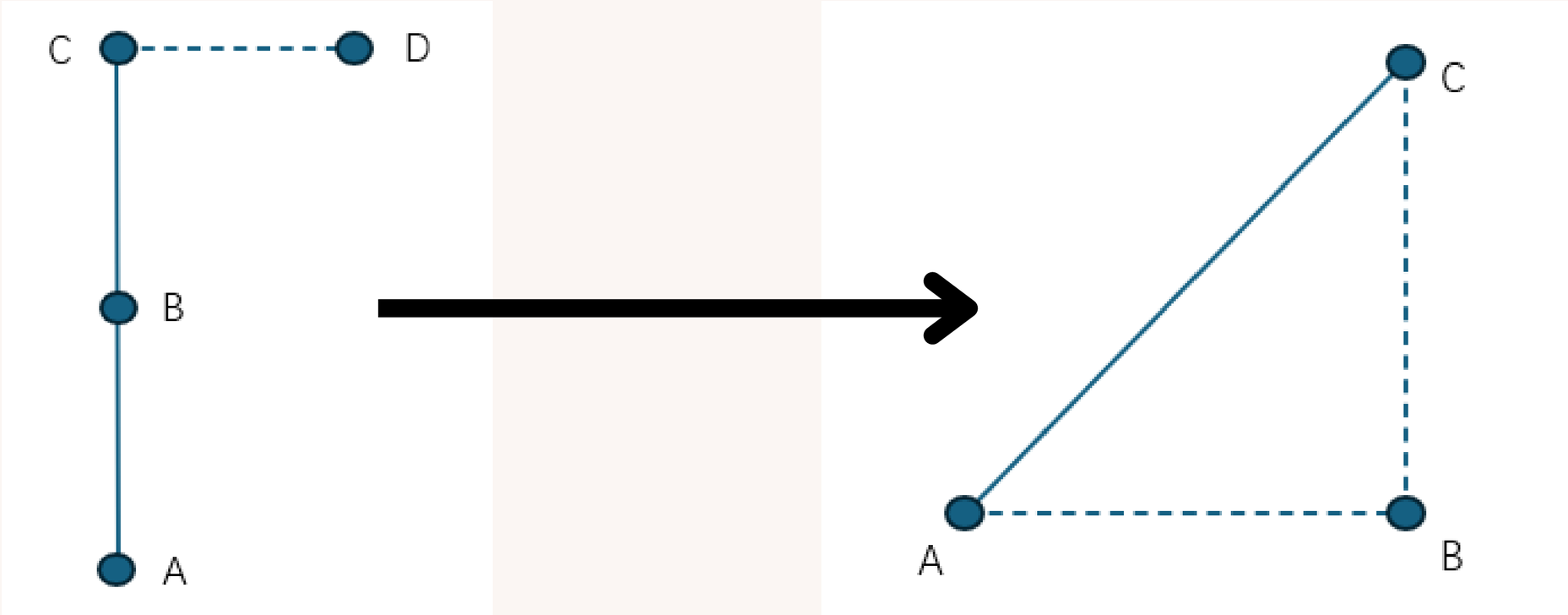

G:2 F:340
G:1 F:306
G:2 F:276
G:1 F:326
G:1
G:4 F:236
G:2 F:316
G:1 F:282
G:3 F:224
G:4 F:276
G:3 F:244
G:5 F:190
G:8 F:168
G:6 F:240
G:5 F:210
G:7 F:160
G:10 F:140
G:8 F:208
G:7 F:180
G:9 F:134
G:12 F:116
G:10 F:180
G:9 F:154
G:11 F:112
G:14 F:96
G:15 F:80
G:16 F:66
G:17 F:54
G:18 F:44
G:19 F:36
G:20 F:30
G:21 F:26
G:22 F:24
G:23 F:24
G:12 F:156
G:11 F:132
G:12 F:112
G:13 F:94
G:14 F:78
G:15 F:64
G:16 F:52
G:17 F:42
G:18 F:34
G:19 F:28
G:20 F:24
G:21 F:22
G:22 F:22
G:14 F:136
G:13 F:114
G:14 F:96
G:15 F:80
G:16 F:66
G:17 F:54
G:18 F:44
G:19 F:36
G:20 F:30
G:21 F:26
G:22 F:24
G:23 F:24
?

Savtsky-Golay smoothing

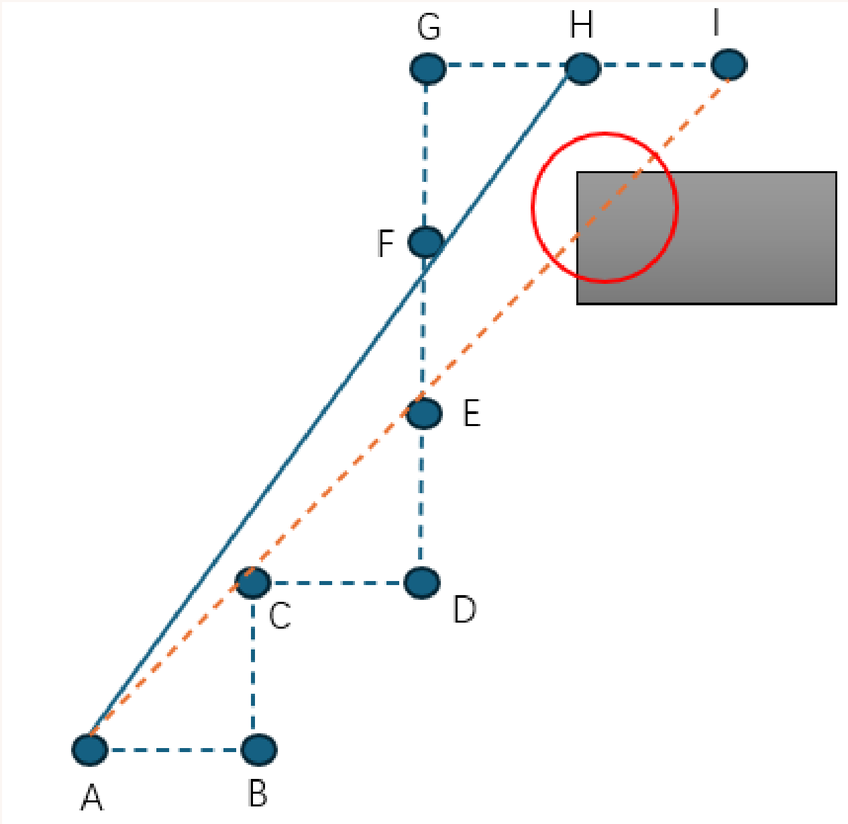
Progressive polynomial fitting

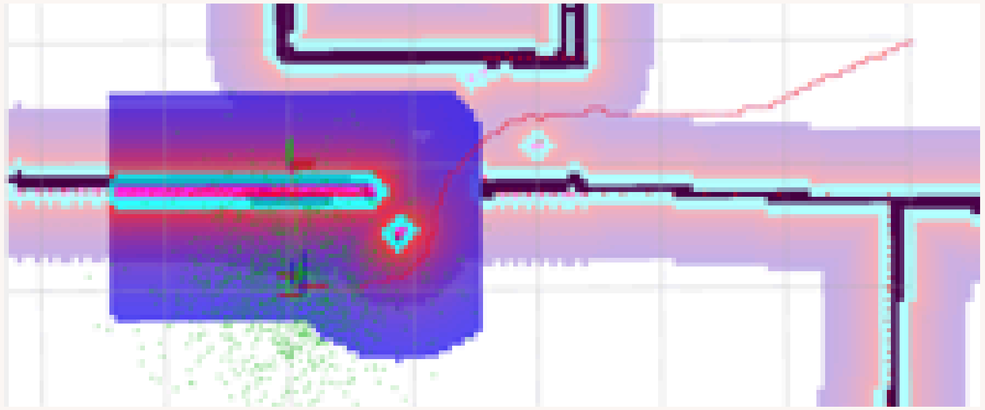


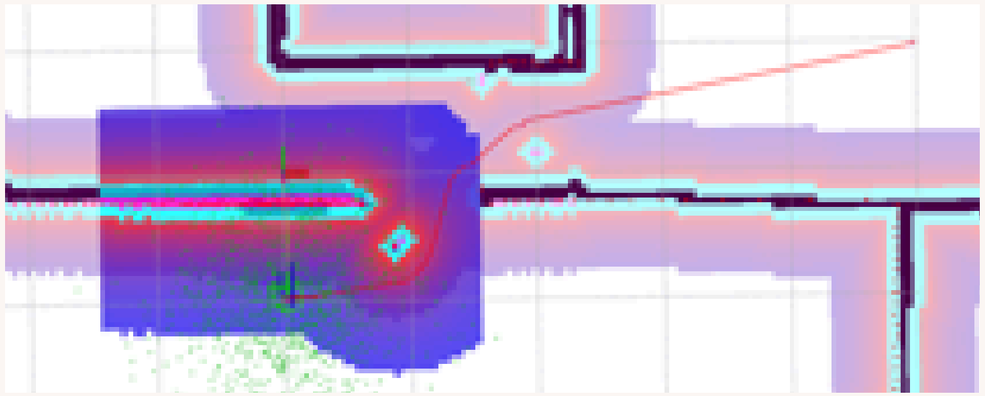
Post-processing



Post-processing





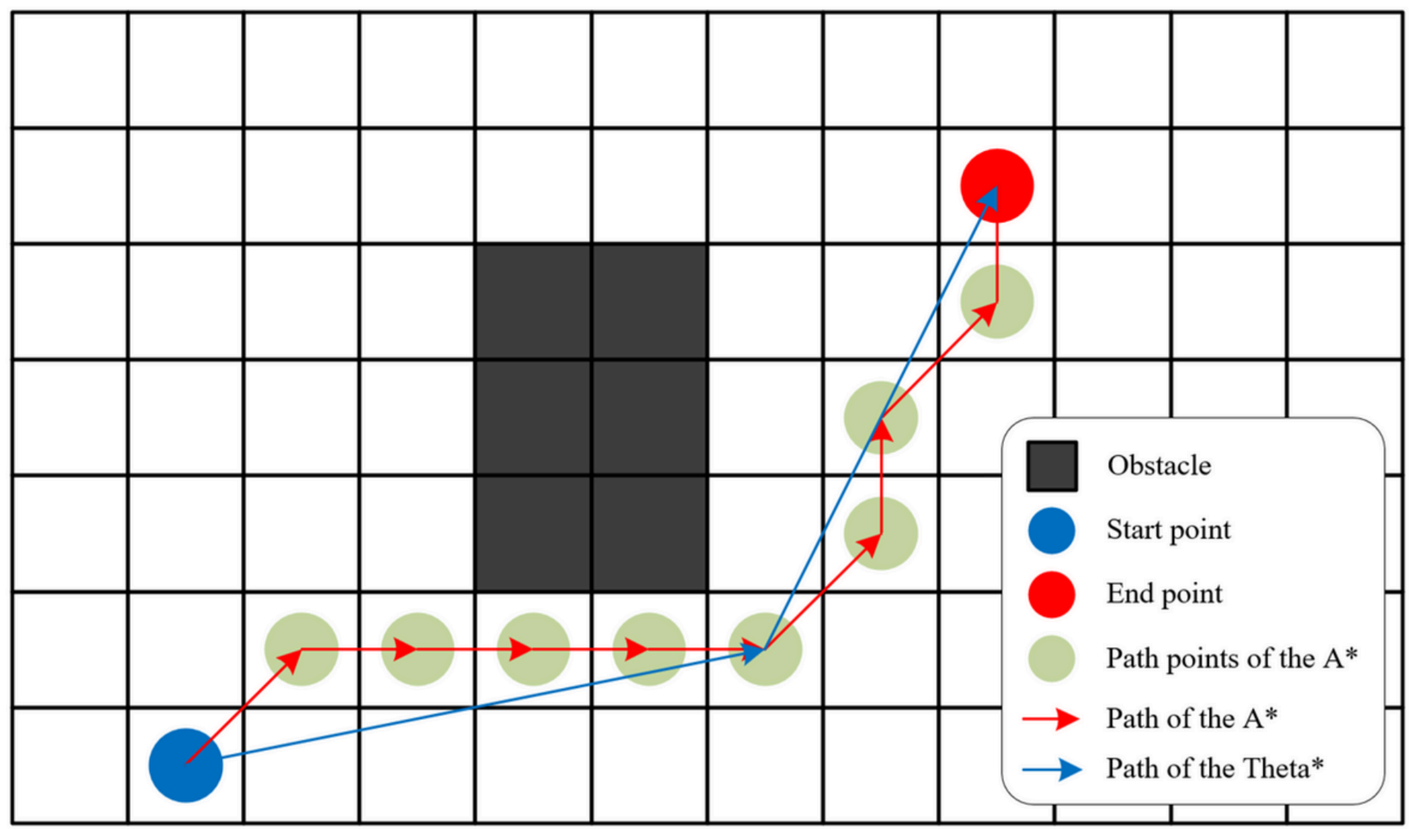


Theta*

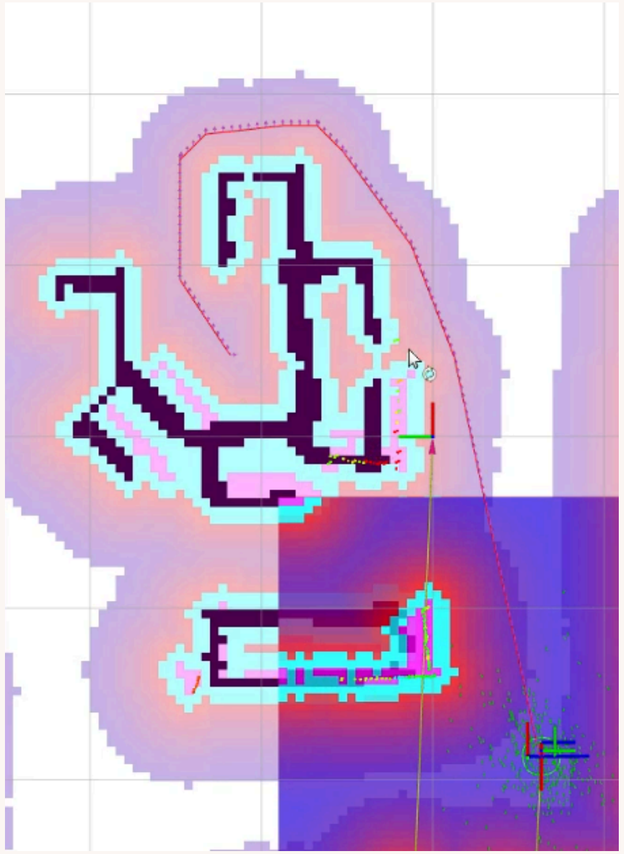
Line of sight check

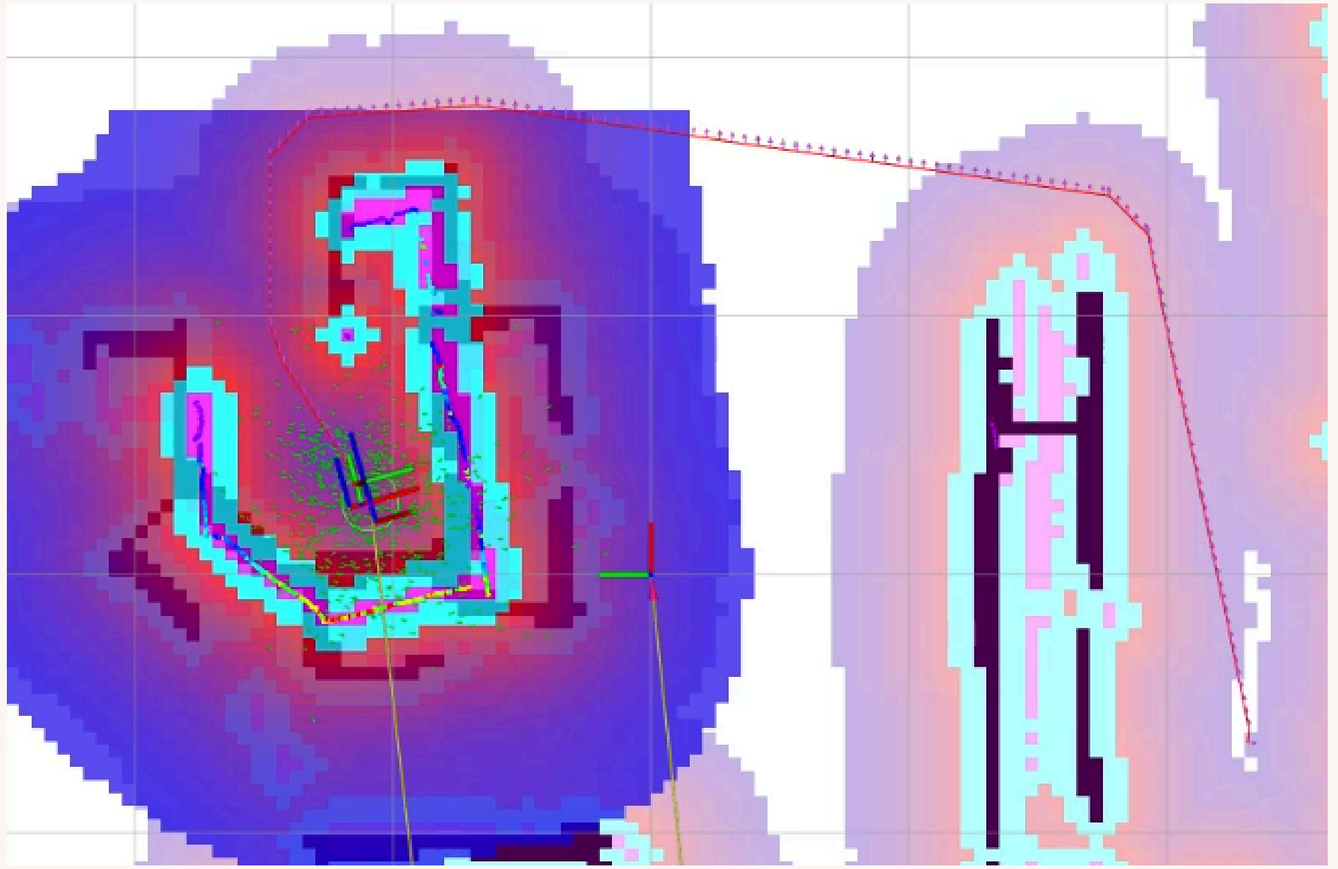
↓

Parent = Parent of neighbor node



Theta*



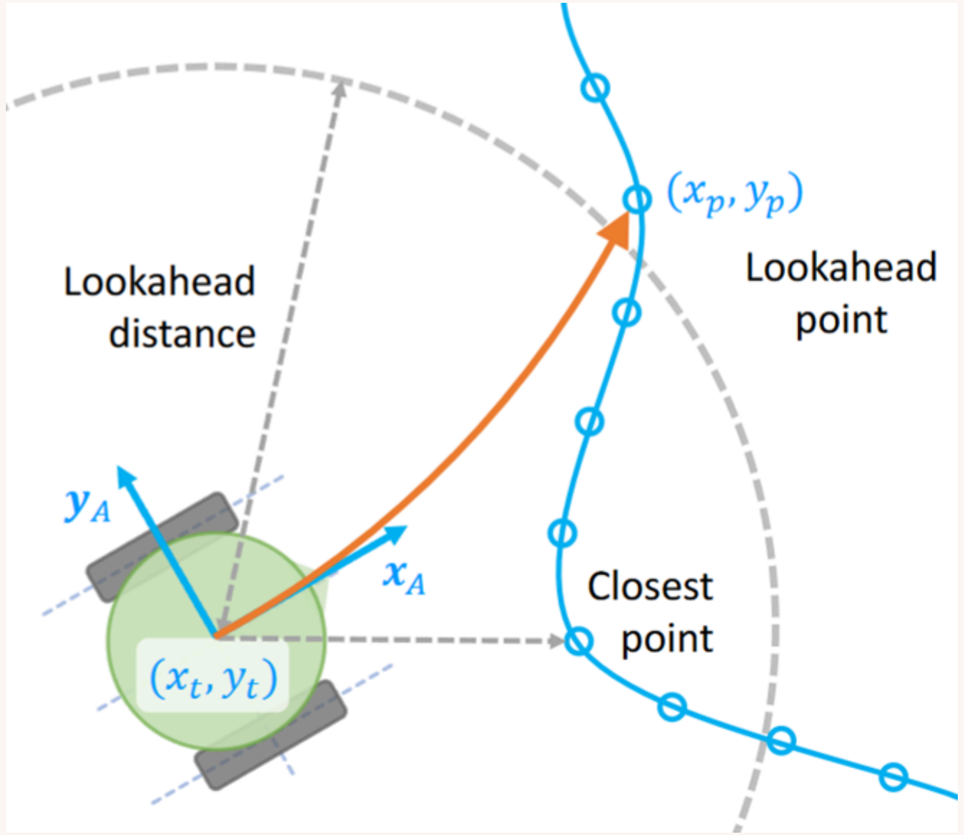


PATH FOLLOWING

PURE PURSUIT

Getting the Lookahead Point

```
// find the lookahead point
int lookahead_idx = closest_idx;
double lookahead_dist = 0.0;
for (int i = closest_idx; i < poses_size; ++i)
{
    lookahead_dist = distToWaypoint(global_plan_.poses[i], pose);
    lookahead_idx = i;
    if (lookahead_dist >= desired_lookahead_dist_)
    {
        break;
    }
}
```

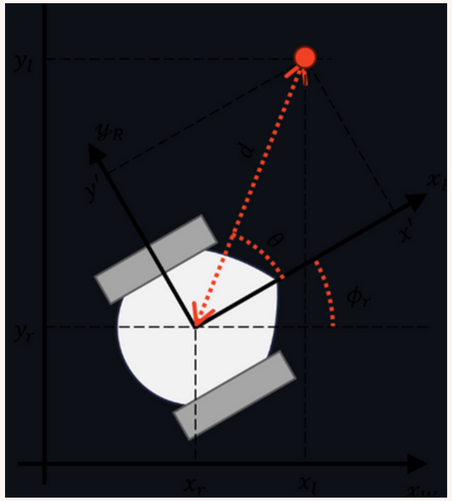


Transform the Lookahead Point to the Robot Frame

```cpp
// transform the lookahead point to the robot frame
geometry_msgs::msg::PoseStamped lookahead_pose_map_frame = global_plan_.poses[lookahead_idx];
double phi = getYawFromQuaternion(pose.pose.orientation);
double dx = lookahead_pose_map_frame.pose.position.x - pose.pose.position.x;
double dy = lookahead_pose_map_frame.pose.position.y - pose.pose.position.y;
double x_prime = dx * std::cos(phi) + dy * std::sin(phi);
double y_prime = -dx * std::sin(phi) + dy * std::cos(phi);
```

$$x' = \Delta x \cos(\phi_r) + \Delta y \sin(\phi_r)$$

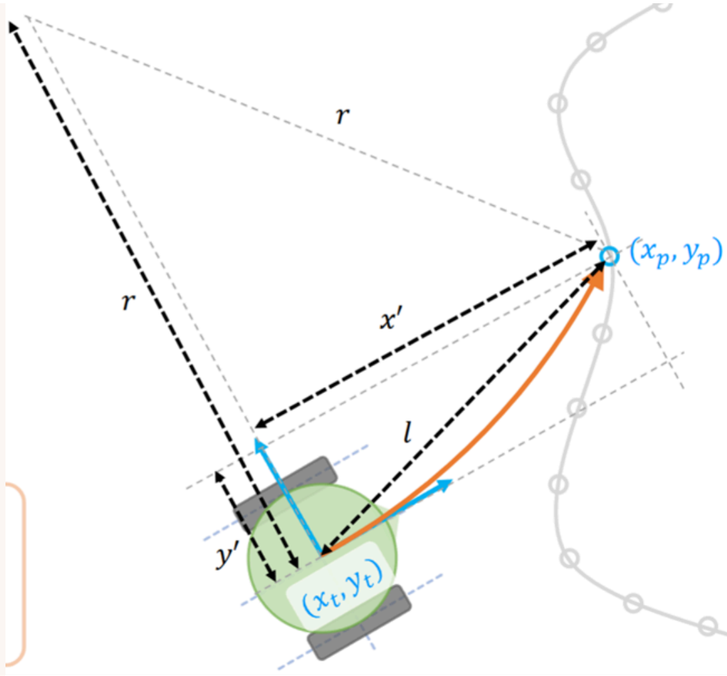
$$y' = \Delta y \cos(\phi_r) - \Delta x \sin(\phi_r)$$



Calculate the Curvature

```cpp
// calculate the curvature c
double L = std::hypot(x_prime, y_prime);
double curvature = 0.0;
const double epsilon = 1e-9;
if (L > epsilon)
{
    curvature = (2.0 * y_prime) / (L * L);
}
```

$$c = \frac{1}{r} = \frac{2y'}{(x')^2 + (y')^2}$$



Calculate Angular Velocity

```
// calculate the angular velocity w
double v = desired_linear_vel_;
double w = v * curvature;
```

$$\omega = vc$$

Constrain Linear and Angular Velocities

```cpp
// constrain linear velocity v by max_linear_vel_
if (std::fabs(v) > max_linear_vel_)
{
    v = std::copysign(max_linear_vel_, v);
}
// constrain angular velocity w by max_angular_vel_
if (std::fabs(w) > max_angular_vel_)
{
    w = std::copysign(max_angular_vel_, w);
}
```

$$v_{\text{temp}}(k) = v(k-1) + v'(k)\Delta t$$

$$v(k) = \begin{cases} \text{sgn}(v_{\max}, v_{\text{temp}}(k)), & \text{if } |v_{\text{temp}}(k)| > v_{\max} \\ v_{\text{temp}}(k), & \text{otherwise} \end{cases}$$

$$\omega_{\text{temp}}(k) = \omega(k-1) + \omega'(k)\Delta t$$

$$\omega(k) = \begin{cases} \text{sgn}(\omega_{\max}, \omega_{\text{temp}}(k)), & \text{if } |\omega_{\text{temp}}(k)| > \omega_{\max} \\ \omega_{\text{temp}}(k), & \text{otherwise} \end{cases}$$

REGULATED PURE PURSUIT

Calculate Curvature Heuristics

```
    double curvatureHeuristic(const double &curvature, double &desired_v, const double &curvatureThreshold)
    {
        if (curvatureThreshold < curvature)
        {
            desired_v = (desired_v * curvatureThreshold) / curvature ;
//            std::cout << "curvature exceeded : " << curvature << std::endl;
        }
        return desired_v;
    }
```

$$v_c = \begin{cases} v' \frac{c_h}{c}, & \text{if } c_h < c \\ v', & \text{otherwise} \end{cases}$$

Calculate Proximity Heuristics

```
    double proximityHeuristic(const double &proximity, double &desired_v, const double &proximityThreshold)
    {
        if (proximity < proximityThreshold)
        {
            desired_v = (desired_v * proximity) / proximityThreshold ;
//              std::cout << "obstacle too close : " << proximity << std::endl;
        }
        return desired_v;
    }
```

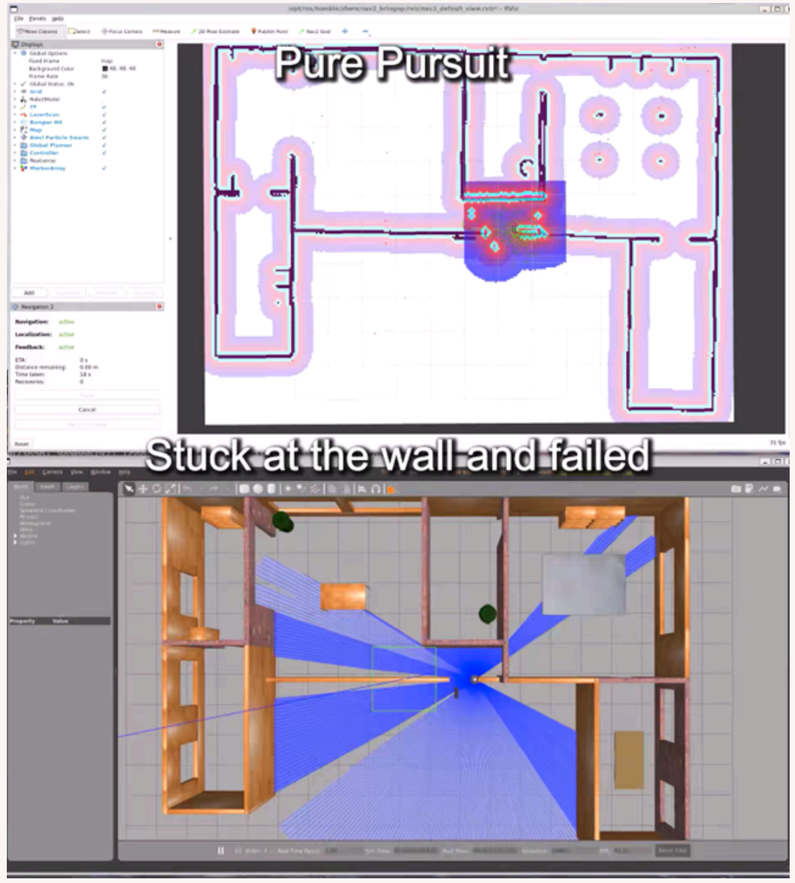
$$v = \begin{cases} v_c \frac{d_o}{d_{\text{prox}}}, & \text{if } d_o < d_{\text{prox}} \\ v_c, & \text{otherwise} \end{cases}$$

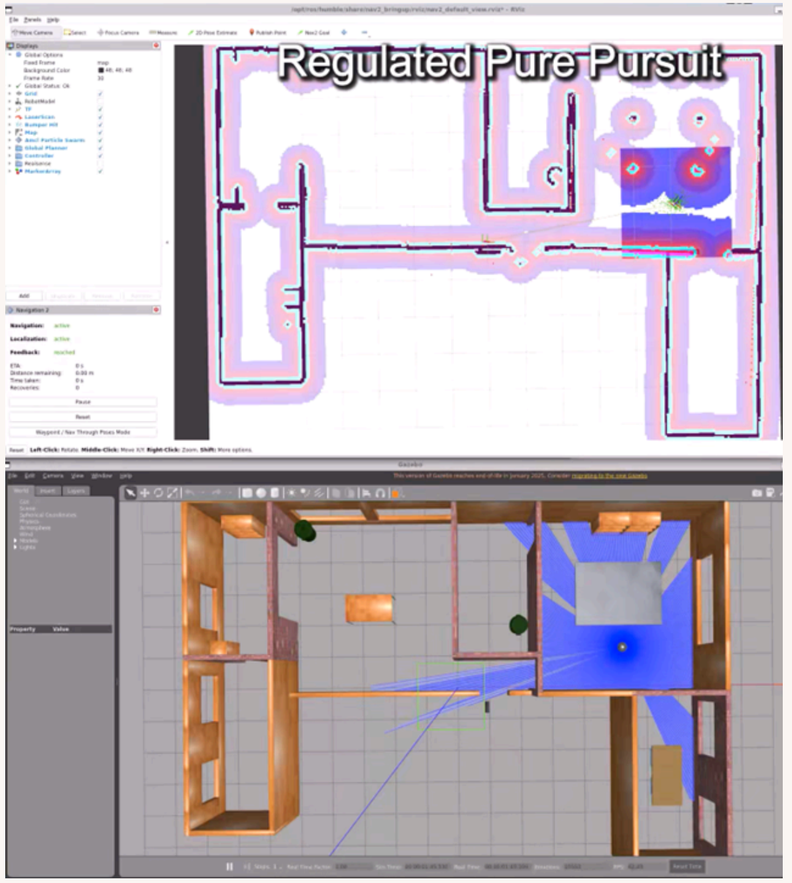
Vary Lookahead

```
double varyLookaheadDistance(const double &lookaheadGain, const double &desired_velocity) {
    return lookaheadGain * desired_velocity;
}
```

$$L_h = v g_l$$

Comparison


Pure Pursuit
Stuck at the wall and failed


Regulated Pure Pursuit

PARAMETER TUNING

Pure pursuit

- Rough intervals were found by dynamically changing the goal points by using the Lab 1 planner or by search on the internet or using AI tools
- Accurate values were found by simulation using the A* planner

Parameters tuned

lookahead gain
desired lookahead dist
proximity threashold
Curvature threashold

Final values :

```
FollowPath:
  plugin: "ee4308::turtle::Controller"
  desired_linear_vel: 0.18
  desired_lookahead_dist: 0.6
  max_angular_vel: 1.0
  max_linear_vel: 0.22
  xy_goal_thres: 0.05
  yaw_goal_thres: 0.25
  curvature_threshold: 0.6
  lookahead_gain: 3.0
  proximity_threshold: 0.25
```

Planners

- Both versions of A* needed the 3 parameters to be tuned because of the use of Savitsky-Golay Smoothing
- Theta* only needed the max_access_cost parameter to be tuned

Parameters tuned

max_access_cost
sg_half_window(not for Theta *)
sg_order(not for Theta *)

Final values :

```yaml
planner_server:
  ros__parameters:
    expected_planner_frequency: 5.0
    use_sim_time: False
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "ee4308::turtle::Planner"
      max_access_cost: 185
      interpolation_distance: 0.1
      sg_half_window: 6
      sg_order: 3
```

Remark

Due to differences between the simulated and physical environments, as well as variations in the robot’s behavior in real-world conditions, most parameters required further adjustments to achieve optimal performance during physical testing.

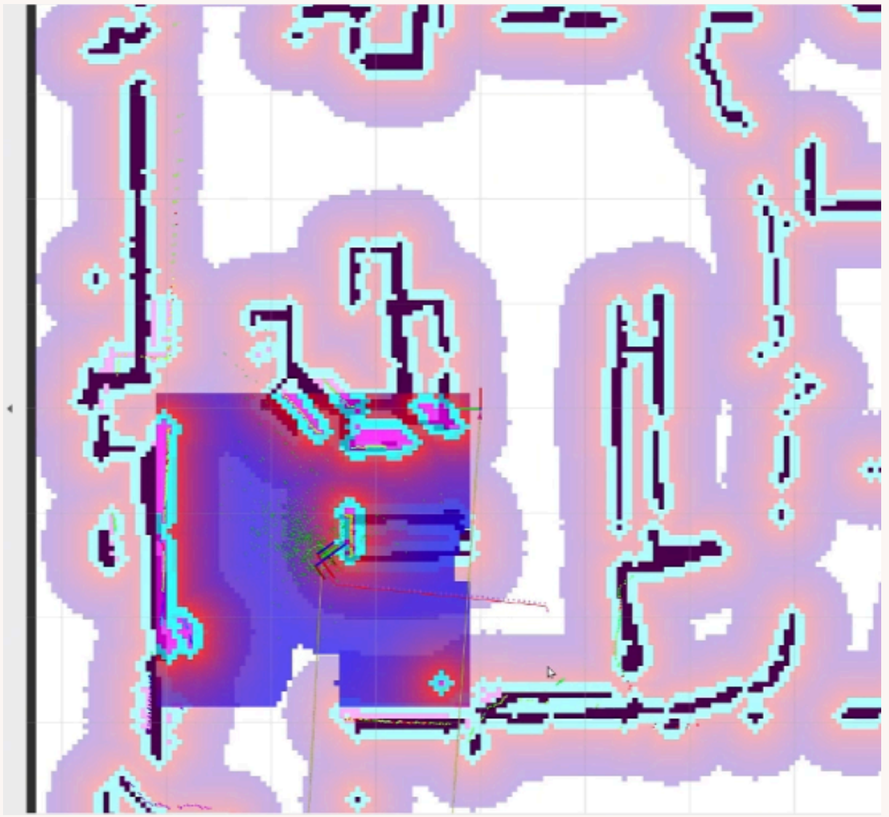
==> The parameters presented were the ones found using the simulation

CONCLUSION FOR CONTROLLER

FINAL LAB RESULTS

Lab Testing

- Real-world conditions with variable lighting, static obstacles, & uneven surfaces
- Metrics: Path efficiency, Obstacle Avoidance & Navigation Time



Key Issues Identified & Improvements

Controller

Problem: Regulated pure pursuit controller could not follow path accurately for lookahead points behind robot

Solution: Added point turning for lookahead points behind robot. Bidirectional control was explored but not used due to instability

Key Issues Identified & Improvements

Planner

Problem: Theta* produced optimal paths but moved too close to obstacles, increasing collision risks

Solution: Multi-cost theta is suggested to incorporate additional cost factors for reducing collision risks

Key Issues Identified & Improvements

Parameter Tuning

Problem: Initial Tuning was ineffective in real-life environments

Solution: New parameter tuning done in lab with multiple iterations

- **Max Access Cost:** Lowered to **175** for safer distance from obstacles
- **Lookahead Distance:** Reduced to **0.6** for better obstacle detection.
- **Goal Thresholds:** Increased to **0.05** (xy) and **0.25** (yaw) to reduce oscillations.
- **Proximity Threshold:** Tuned to **0.10** to balance obstacle avoidance & path availability.
- **Curvature Threshold:** Adjusted to **2.5** to prevent excessive slowing near obstacles.

Final Evaluation

Controller: Regulated Pure Pursuit with point turning works best for differential drive robots

Planner: Multi-cost Theta* should be implemented over Theta* for safer and more efficient path planning

Extensions: Test robot with dynamic obstacles

THANK YOU!