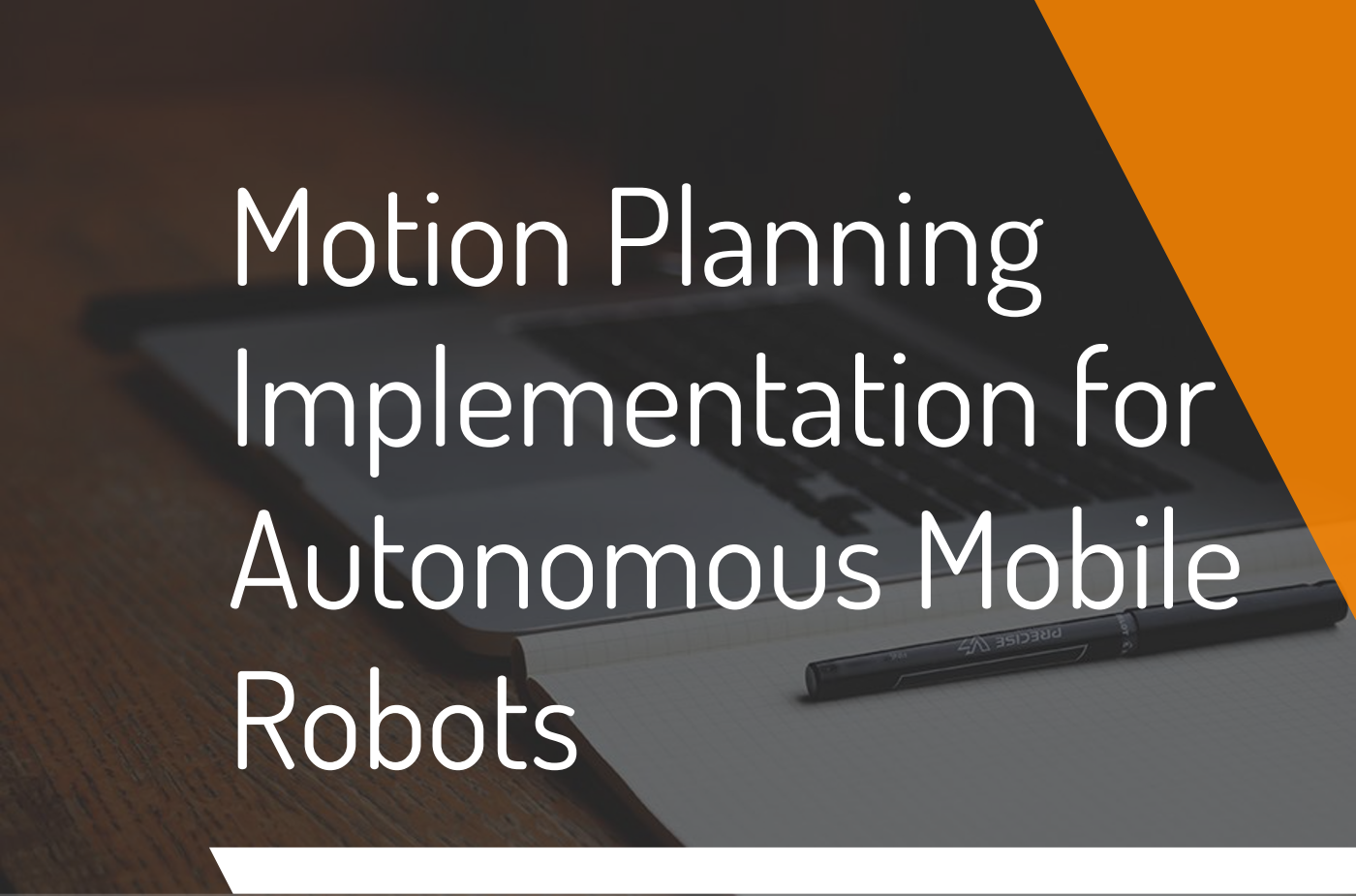
Motion Planning Implementation for Autonomous Mobile Robots



Sachin Sulkunte

University of Maryland

B.S. Computer Engineering

Minors:

- Robotics & Autonomous Systems
- Cybersecurity

1.
Previous Technical Experience

Prior Work Experience

Internship Experience:

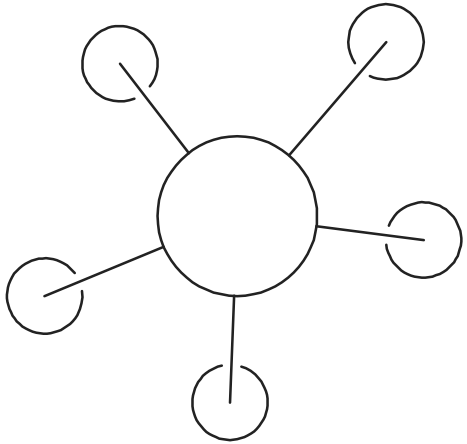
- Shield AI
  - *Python, C++, Robot Operating System, Linux*
- Stryker
  - *C++, Qt, MATLAB, Linux*
- National Institute of Standards and Technology
  - *Python, Computer Vision, Hardware Integration*
- Praxis Engineering (General Dynamics)
  - *Python, AWS, Databases, Computer Vision*

Prior Personal Projects

Project Experience:

- UMDLoop: Undergraduate Engineering Team
  - *Embedded C, Sensor Integration, Networking, etc.*
- VR-Controlled 6DOF Robot Arm
  - *Simulation, ROS, Kinematics, C++*
- Covid19 AI Chatbot and Mask Detector
  - *Python, LLM, Computer Vision*

2. Project Overview



Project Focus

Motion Planning for an Autonomous Mobile Robot

Autonomous Mobile Robot (AMR)

Motivation

- Highly complex, scalable system to design and build
- Facilitates development of variety of intricate software features
  - Machine Learning
  - Planning and Controls
  - Embedded Systems
  - Networking
  - And more!

AMR Development Cycle – High Level

Project Scope – System Requirements

Shall…

- run on NVIDIA Jetson Nano SBC
- determine collision-free paths to waypoint
- navigate autonomously given known map

Should…

- use Robot Operating System (ROS)
- be capable of mapping unknown spaces with unknown features
- retain localization data to ~1cm accuracy

Extra Features

- Deep learning models
- Model predictive control (MPC)
- Visual odometry

Technologies Used

C++/C

C++ for ROS node development

C for basic Arduino development

Python

Used for computer vision related tasks including training custom models

ROS

Utilized for hardware abstraction, inter-process communication, testing, and visualization

CUDA

Speed up processing to permit fast 3D computations for path planning, SLAM, and visual odometry

Git

Utilized for effective version control including developing features on branches and merging to master

Gtest

ROS implementation of GoogleTest framework for C++ unit testing

3.
Path Planning

Path Planning Algorithms

	Pros	Cons	Example
Grid-Based	Simple and effective	Difficult to determine heuristic function	A*
Sampling-based	Well-adapted to dynamic environments	Not guaranteed best solution	RRT
Machine Learning Optimization	Modify policy based on env feedback	Highly-complex optimization problem	Markov Decision Processes

Rapidly Exploring Random Trees (RRT)

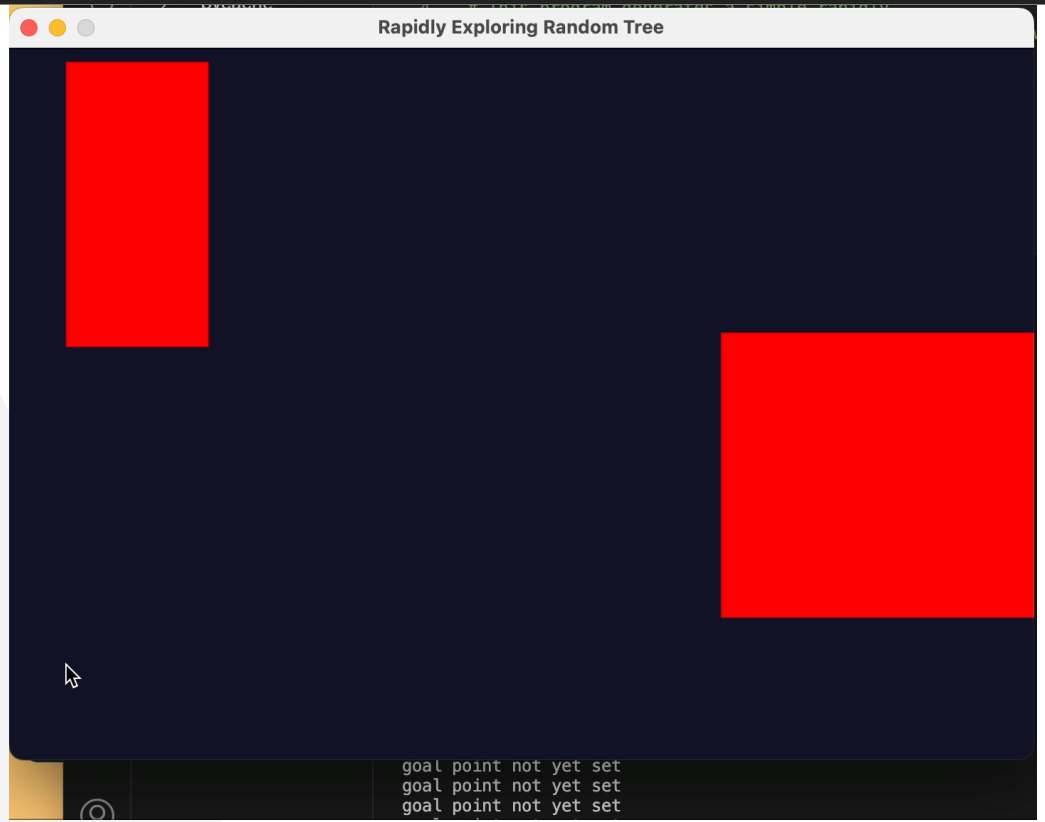
Ove

- R
- C mples
  ir
- In areas
  o

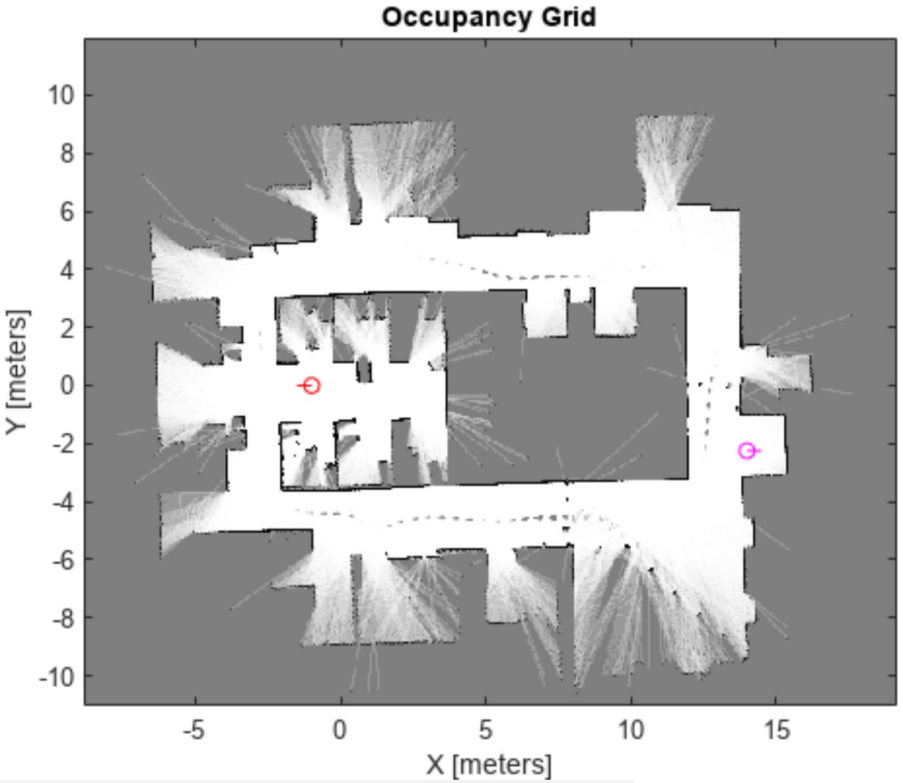
RRT

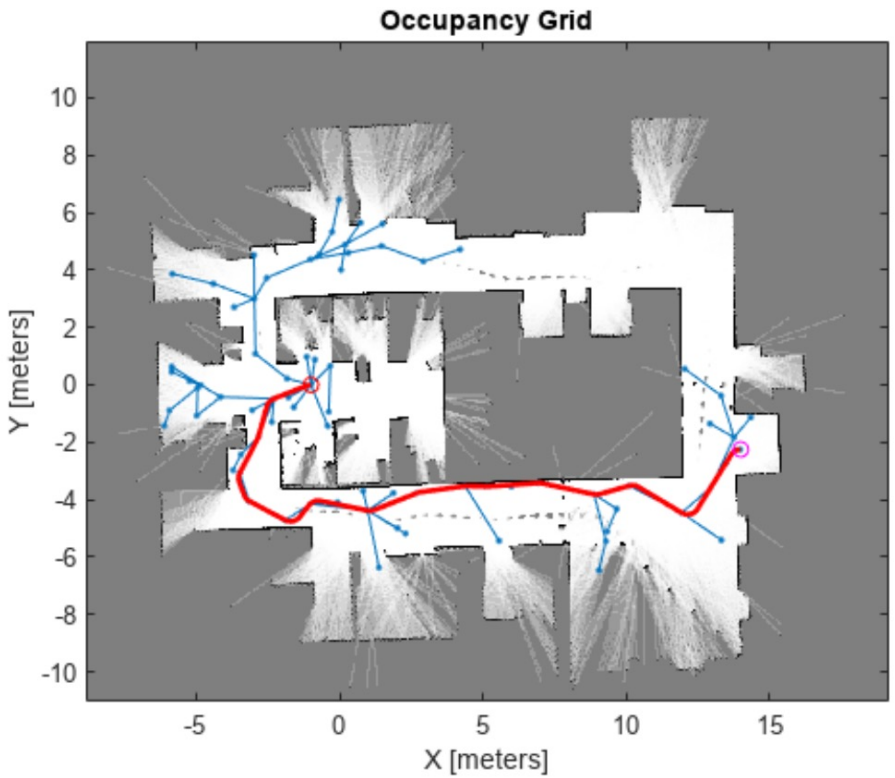
```
function RRT(start, goal, obstacles, step_size):
    tree = create_tree(start)

    while (goal.found == false):
        random_point = generate_random_point()
        nearest_node = find_nearest_node(tree, random_point)
        new_node = steer(nearest_node, random_point, step_size)

        if not collides_with_obstacles(nearest_node, new_node, obstacles):
            add_node_to_tree(tree, new_node)

            if is_near_goal(new_node, goal, step_size):
                final_path = construct_path(tree, new_node)
                return final_path

    return None  # No path found within imposed constraints
```

RRT Simulation Implementation



Further Implementations





Problem: Non-Optimal Route

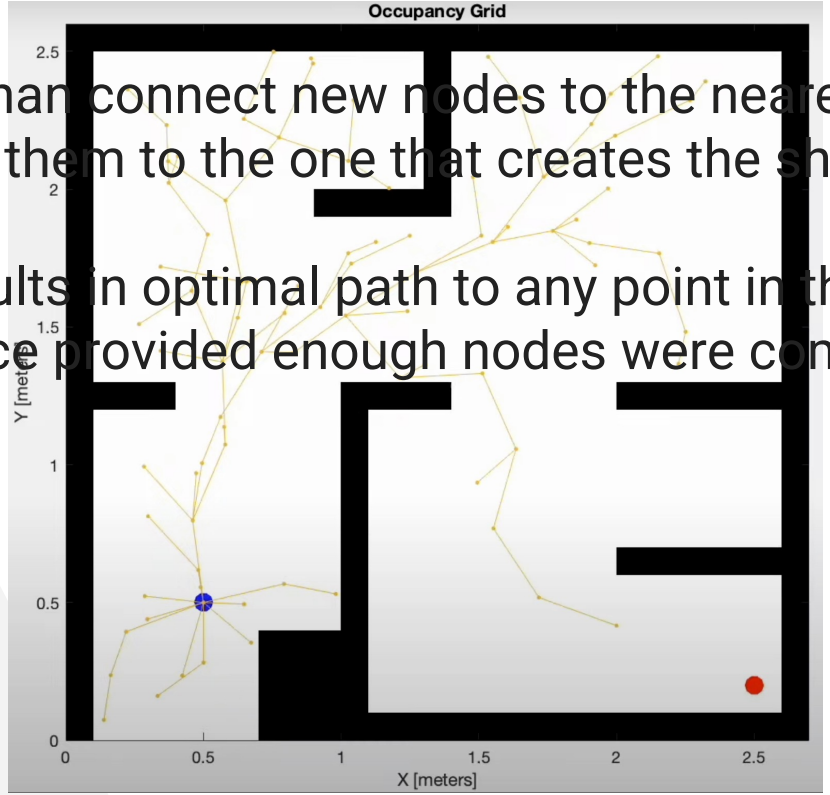
- RRT generates a very jagged and non-optimal path to the goal point
  - On a differential-drive robot, this results in
    - significant loss of speed due to minimal acceleration time
    - accuracy loss due to frequent stops and turns

Solution:

RRT* Algorithm

RRT* Algorithm

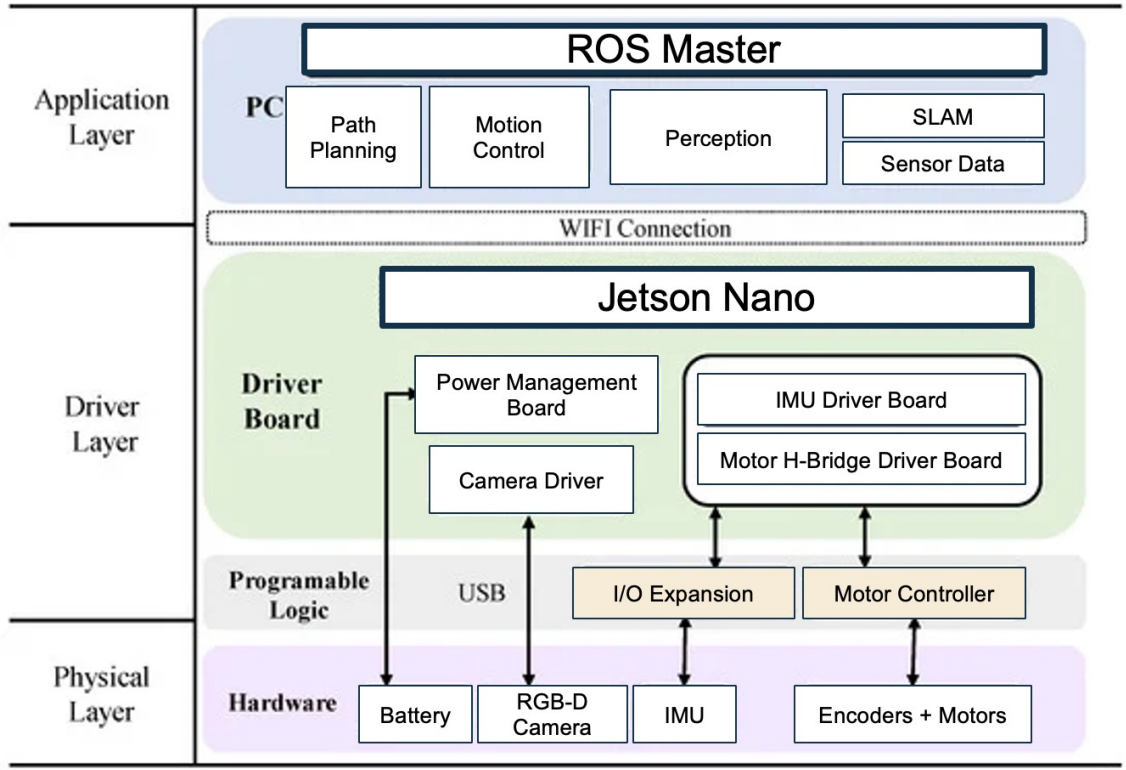
- Rather than connect new nodes to the nearest node, connect them to the one that creates the shortest path to start
  - Results in optimal path to any point in the search space provided enough nodes were computed



Translating to Robot

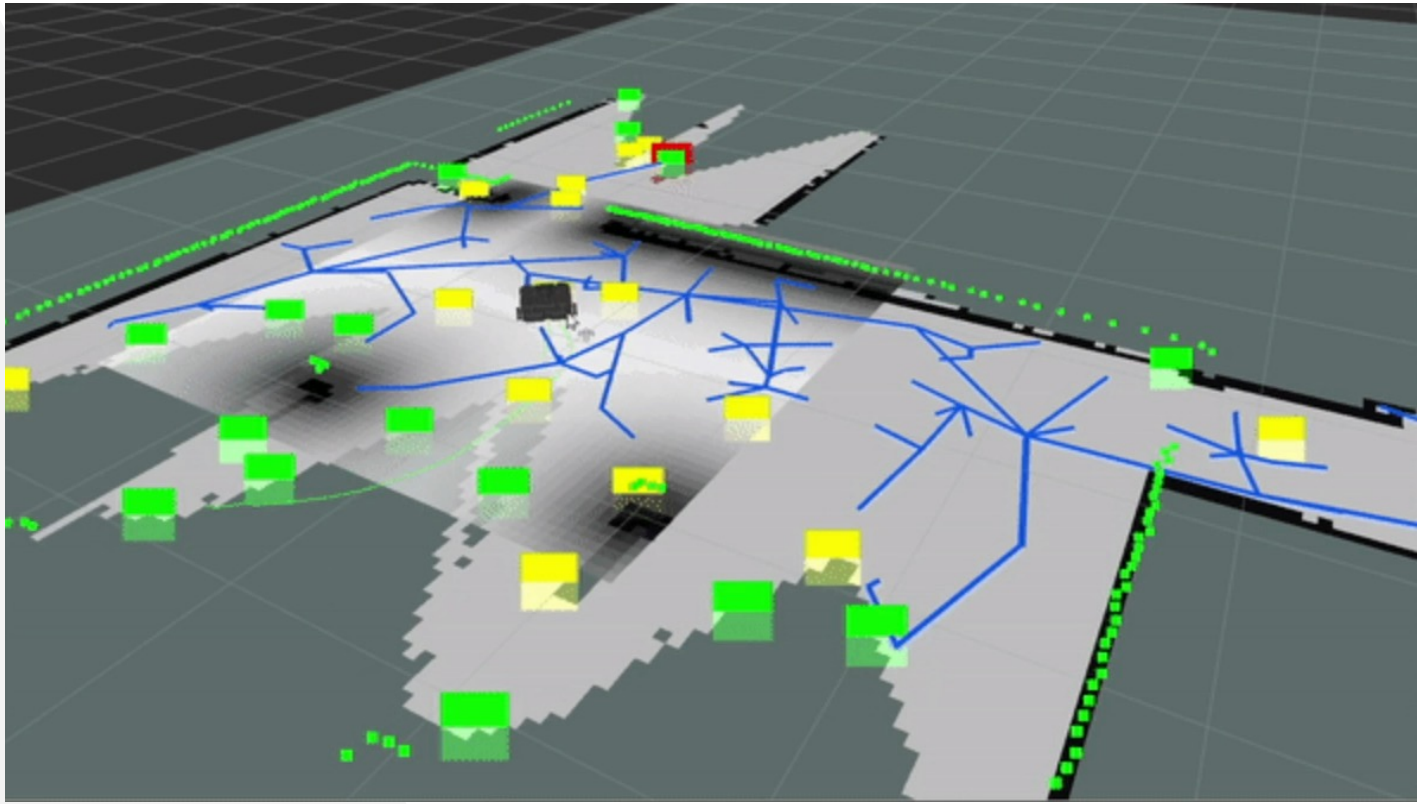
- Algorithm only considers two states
  - X-position
  - Y-position
- Must consider orientation ϕ
  - Robot can only move along its orientation axis
- System functions by:
  - Computing path planning in ROS node based on map
  - Path positions are passed to a node which publishes needed wheel velocities
  - Wheel velocities are sent to the controller with PID tuning feedback

Motion Control

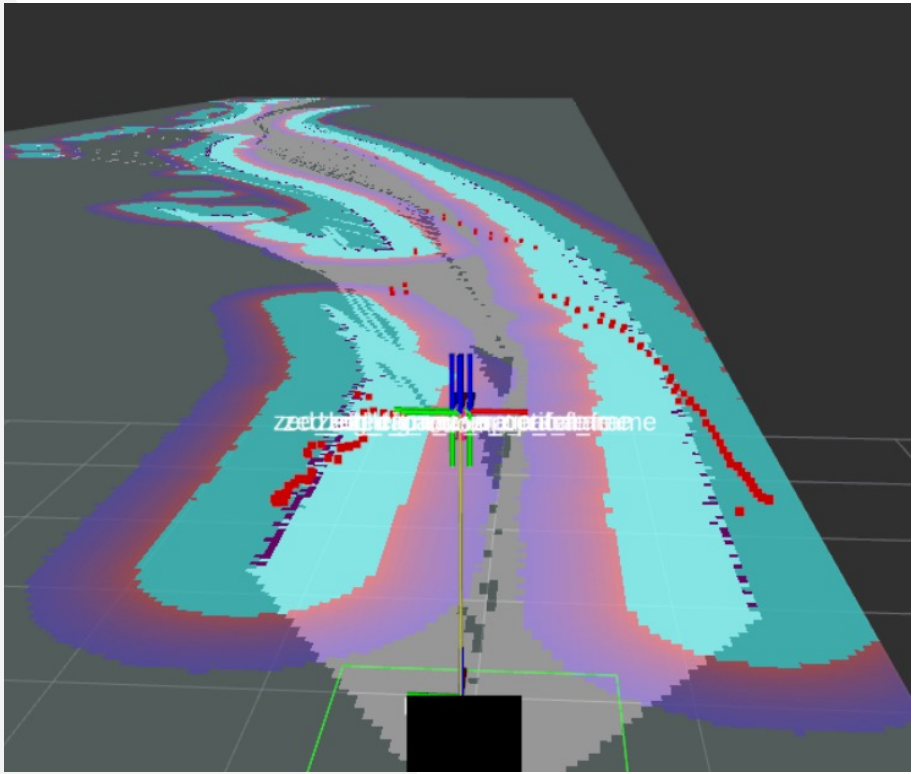


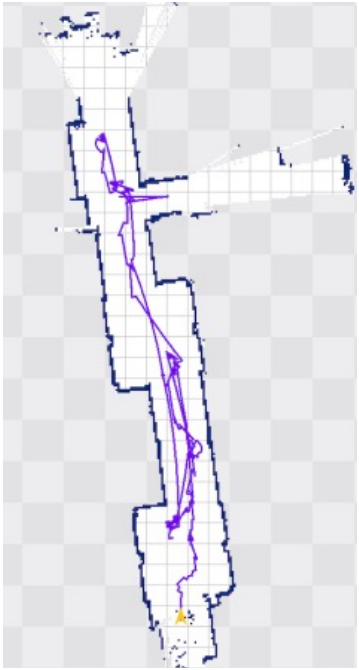
4. Testing / Verification and Results

Simulation Testing in Gazebo



Further Testing





Resulting occupancy grid

Unit Testing in C++

- Gtest framework implemented in ROS for unit and component testing
- Created permanent data objects using test fixtures for complex, repeat testing

```
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from RRTServer
[ RUN      ] RRTServer.loadValidMapConfig
[       OK ] RRTServer.loadValidMapConfig
[ RUN      ] RRTServer.computeValidNodes
[       OK ] RRTServer.computeValidNodes
[ RUN      ] RRTServer.verifyOptimalPath
[       OK ] RRTServer.verifyOptimalPath
[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran.
[  PASSED  ] 3 tests.
```

Lessons Learned

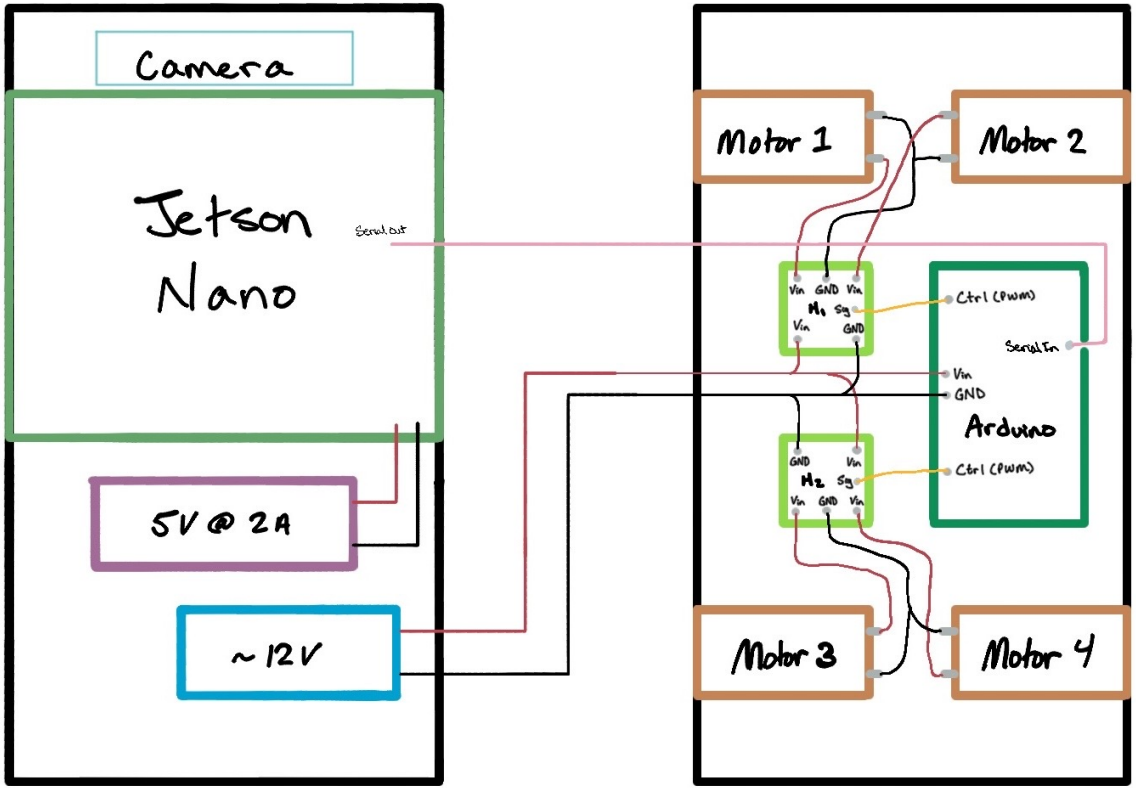
- Differential-drive systems are highly-susceptible to physical disturbances leading to ~5cm error in accuracy over 5m of travel distance
- Maintaining clean data flow in the system
  - Complex array of publishers and subscribers with data requirements
- Necessity of optimizing algorithms for speed
  - Limited computation power in an onboard SBC
- Challenges of sim-to-real transition
  - Ideal vs non-ideal system, hardware configuration issues

THANKS!

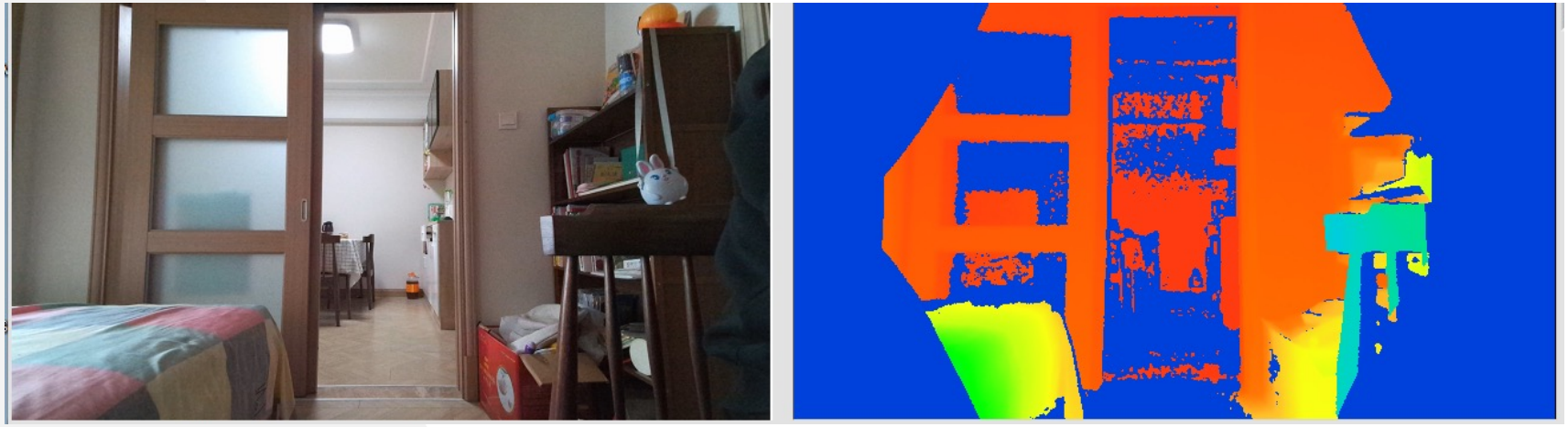
Questions?

Appendix

Basic Hardware Design



Perception Input: RGB-D Image



Kinect Images: RGB, Depth Map