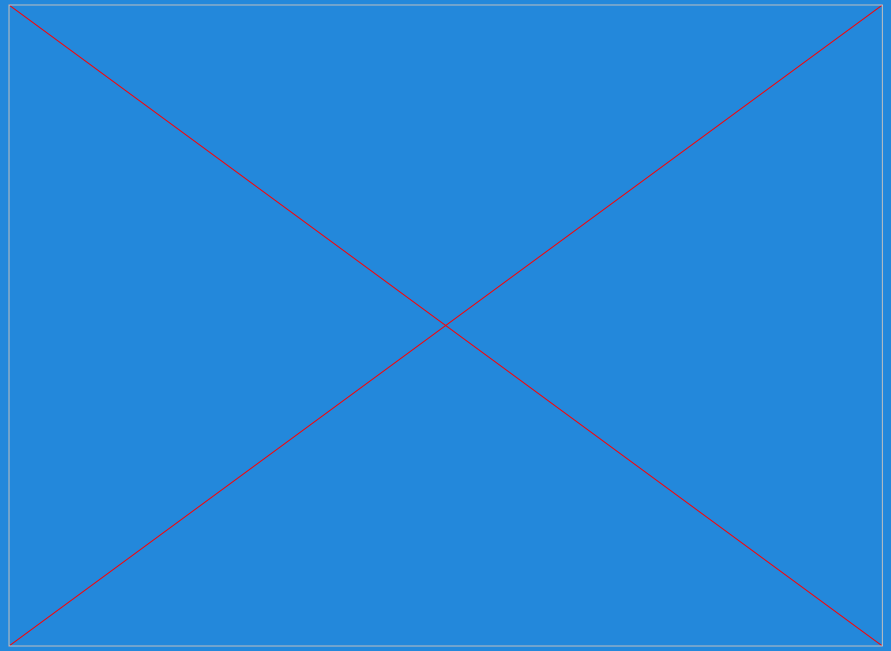
Agribot



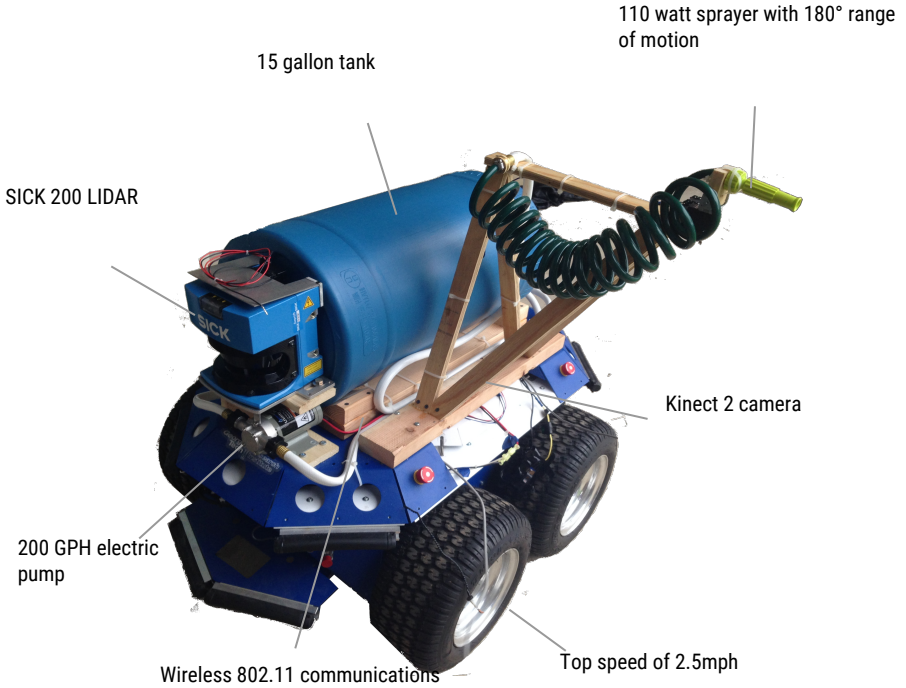
Sprayer, SLAM, and Robust Navigation

Andrey Kurenkov, Troy O'Neal, Pavel Komarov

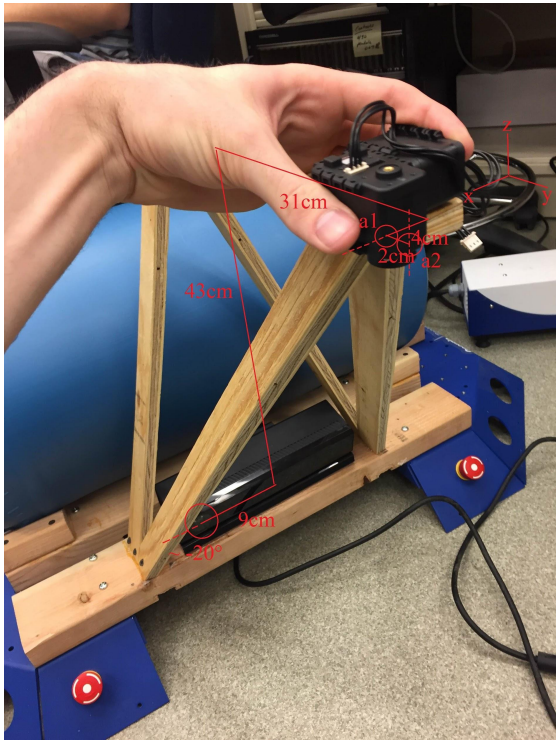
Agribot Robot Design

Problem Statements:

1) SLAM (localization, mapping, and plant detection)
2) Plan a path to goal location, avoid obstacles
3) Design and aim of liquid sprayer



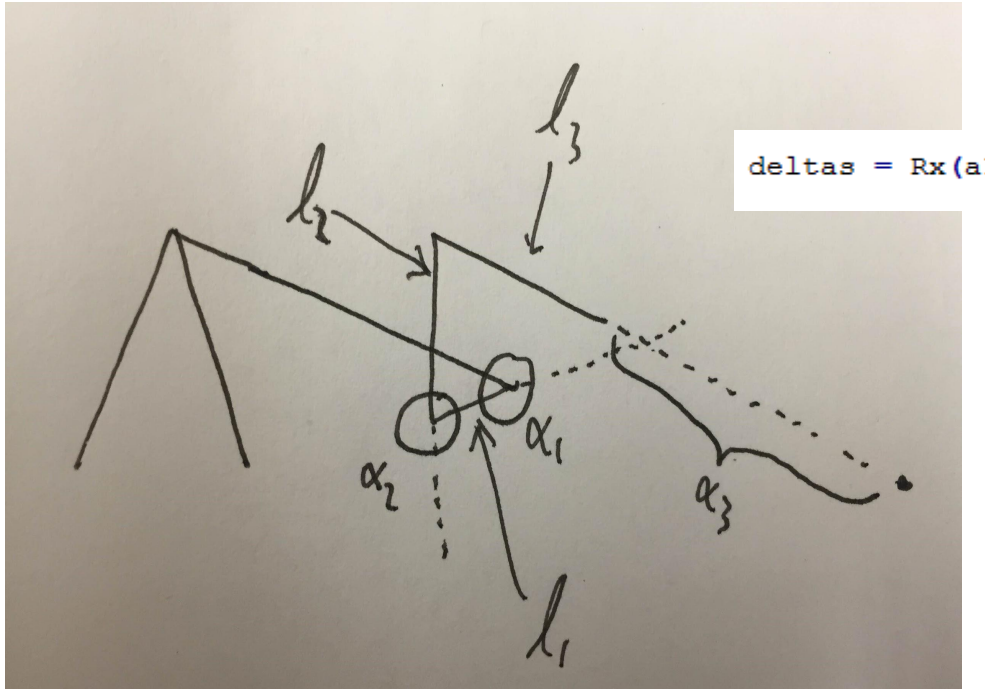
Sprayer Design



Initial Sprayer Layout

Final Sprayer Assembly

Sprayer Inverse Kinematics

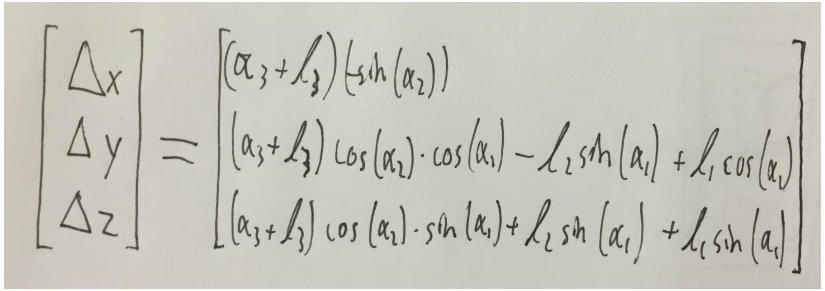



```
deltas = Rx(a1)*Rz(a2)*[0; l3+a3; l2] + Rx(a1)*[0; l1; 0]
```

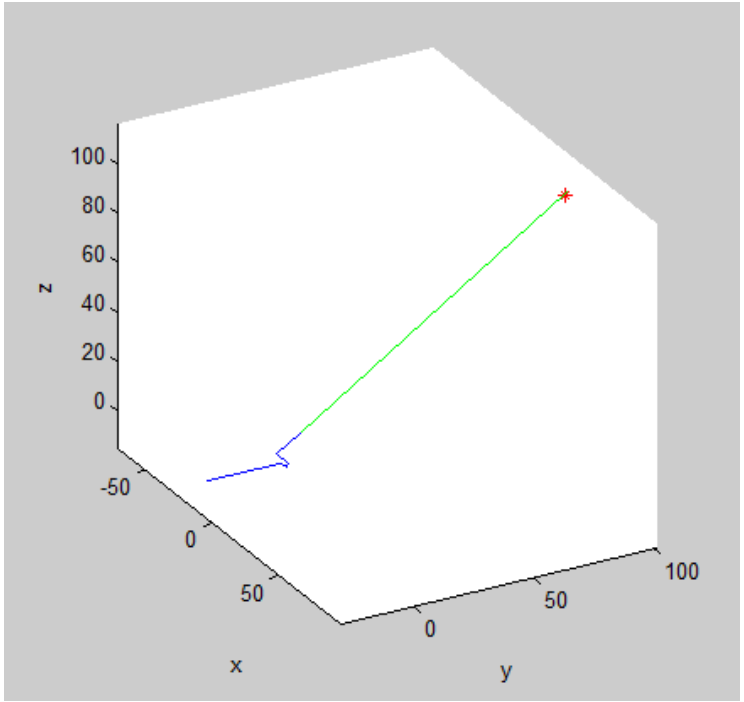
```
deltas =

                                          -sin(a2)*(a3 + l3)
 l1*cos(a1) - l2*sin(a1) + cos(a1)*cos(a2)*(a3 + l3)
 l2*cos(a1) + l1*sin(a1) + cos(a2)*sin(a1)*(a3 + l3)
```

Sprayer Embedded Control

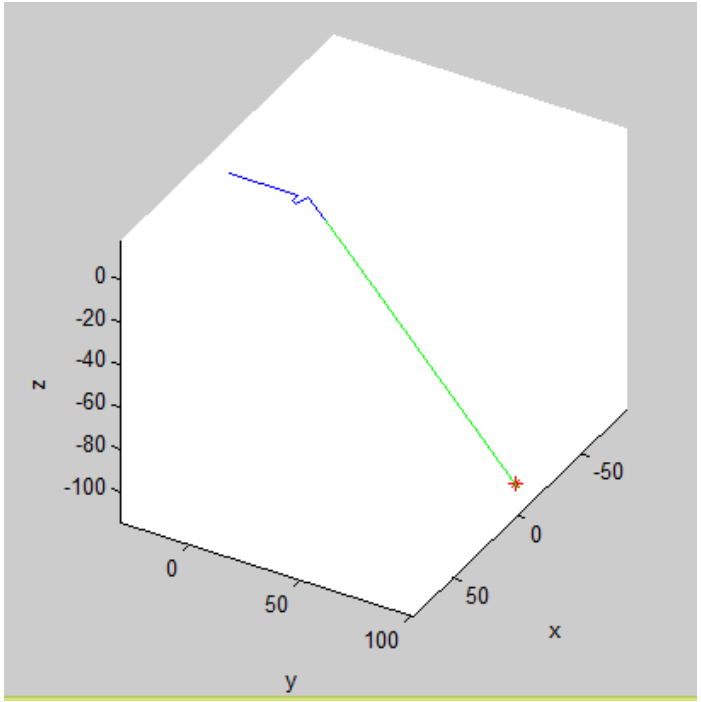
```
//Convert spray coordinates in to joint angles
void get_coords(spray_data *spray){
    float dist, den, lx, num;
    float angles[] = {0.0,0.0};
    float dx = spray->x - X_OFFSET;
    float dy = spray->y - Y_OFFSET;
    float dz = spray->z - Z_OFFSET;

    //Use relationships derived on paper to calculate alphas in terms
    //of these deltas and lengths
    dist = sqrt(dx*dx + dy*dy + dz*dz - L1*L1 - L2*L2) - L3;
    angles[1] = asin(-dx/(dist+L3)) *180/PI;

    lx = (dist+L3)*cos(angles[1]*PI/180)+L1;
    num = (dz>0?-1:1)*sqrt( ((dy/dz)*(dy/dz) + 1) *(lx*lx + L2*L2)) + lx*(dy/dz) + L2;
    den = lx - L2*(dy/dz);

    angles[0] = (-2*atan(num/den))*180/PI;

    spray->phi_2=150+(int)angles[0];
    spray->theta_1=150+(int)angles[1];
}
```



Sprayer IK Simulation

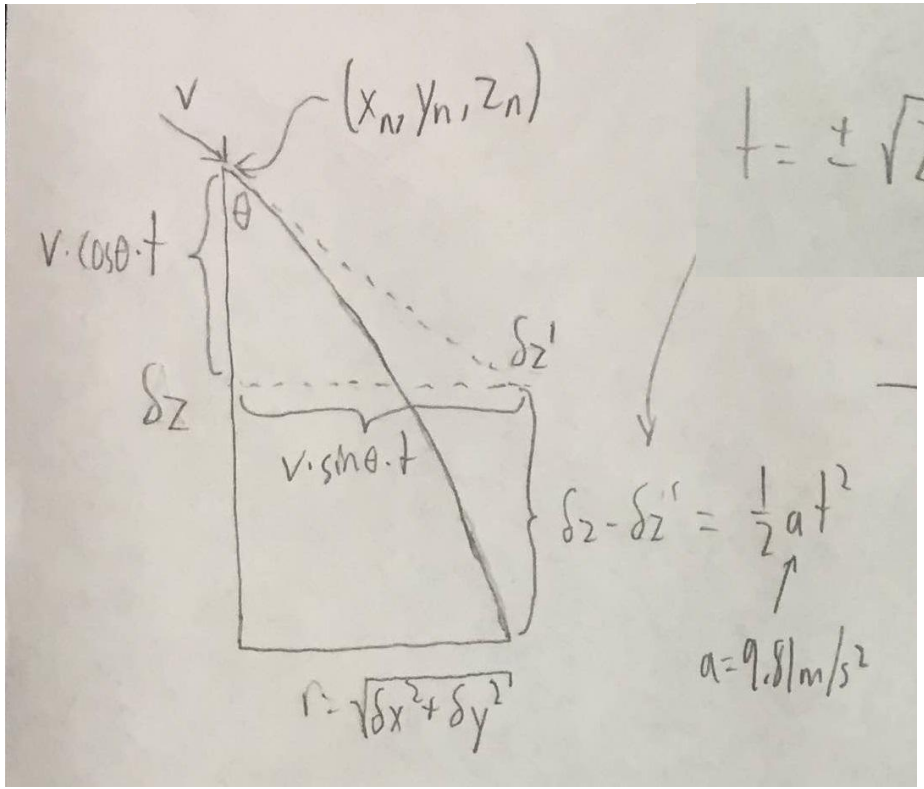


(30, 100, 100)

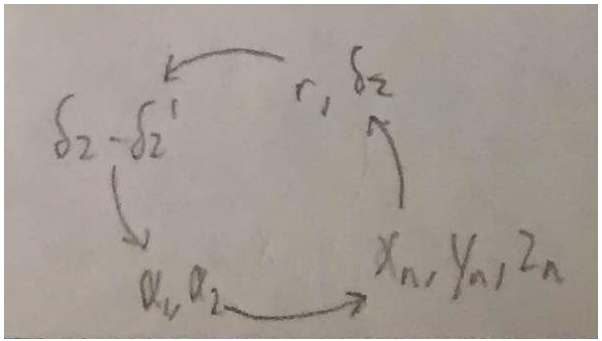


(0, 100,- 100)

Gravity-Compensating IK



$$t = \pm\sqrt{2}\sqrt{\frac{\pm\sqrt{-a^2r^2 + 2av^2\delta z + v^4}}{a^2} + \frac{v^2}{a^2} + \frac{\delta z}{a}}$$



SLAM and Plant Detection

Problem:

Simultaneous

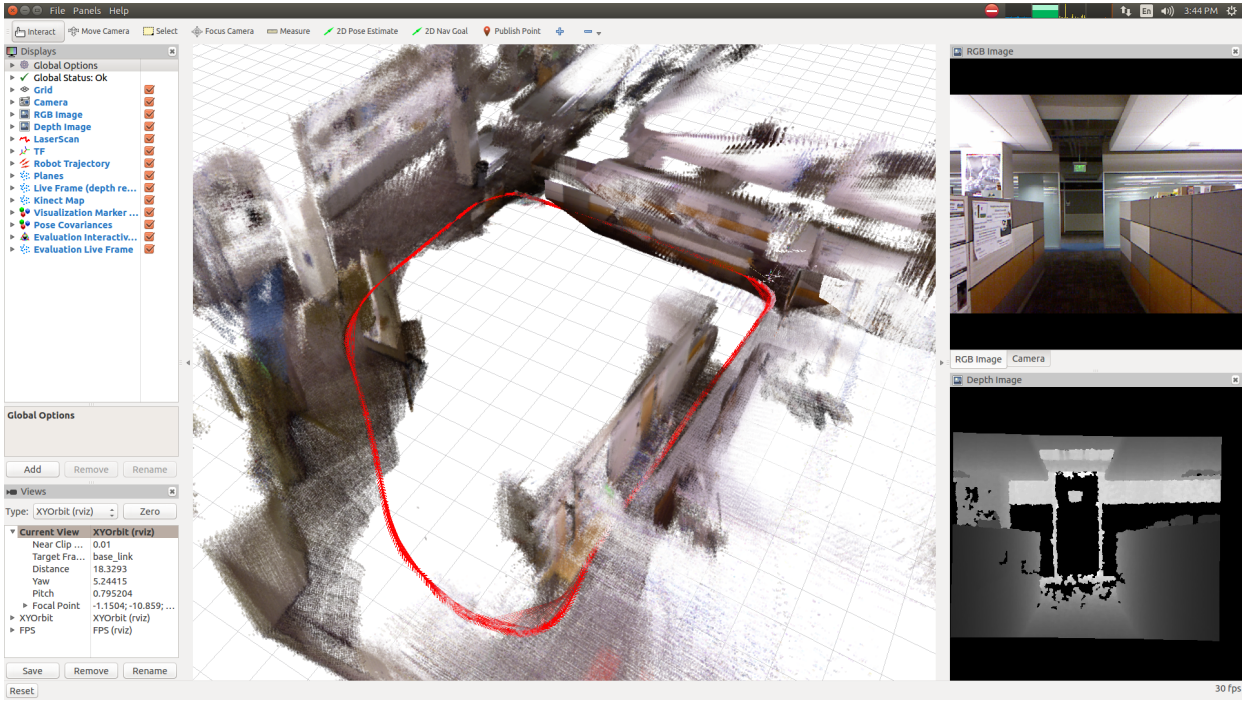
Localization

and Mapping with

plant detection

Want to combine:

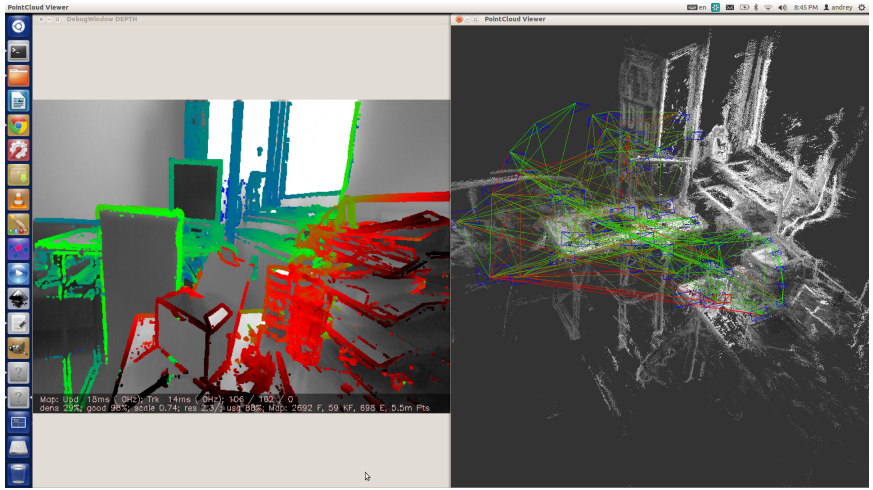
- Kinect 2
- Odometry
- LIDAR



Visualization of SLAM from OmniMapper

Summary of ROS-based SLAM

- RatSLAM
  - Bio-inspired SLAM
  - Combines of monocular images and odometry
- LSD-SLAM
  - Purely monocular SLAM
  - Uses direct image alignment
- RGBD SLAM (V2)
  - Uses RGBD (RGB-Depth) data
  - Uses the RBG feed with RANSAC
- RTAB-Map
  - Builds on RGBD SLAM
  - Adds support for multi-session and large-maps
- MonoSLAM
  - Monocular SLAM, standard 1-point RANSAC with an Extended Kalman Filter for motion
  - Inverse depth parametrization to get the 3D point locations for mapping.



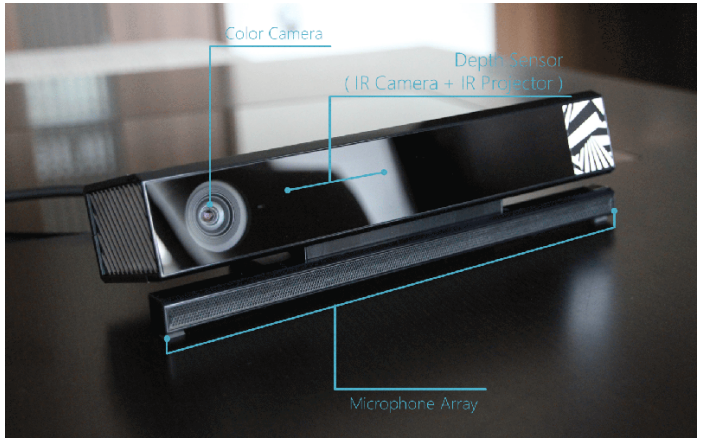
lsdSLAM demo output

OmniMapper

- OmniMapper is a framework for SLAM
  - A plugin-based architecture; allows different sensor types to be combined for SLAM.
  - The only real ROS-based SLAM framework for sensor fusion
  - "The key contribution is an approach to integrating data from multiple sensors ... so all measurements can be considered jointly."

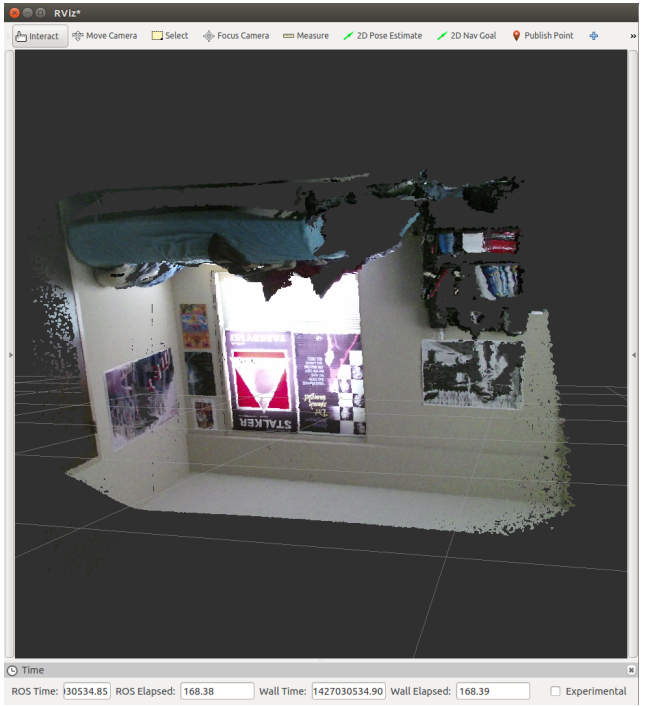
- OmniMapper has the “backend” of Square Root Smoothing And Mapping
  - GTSAM implements the SLAM by optimizing the robot trajectory and landmark positions with a factor graph-based approach
  - The factors can be different sensors or other variables
  - Rather than optimizing just for the latest pose measurement the "smoothing" part of the approach means that the entire trajectory is continually optimized with new input.

Kinect 2



Feature	Kinect for Windows 2
Color Camera	1920 x 1080 @30 fps
Depth Camera	512 x 424
Max Depth Distance	~4.5 M
Min Depth Distance	50 cm
Horizontal Field of View	70 degrees
Vertical Field of View	60 degrees
Tilt Motor	no

- Kinect 2 has is an RGBD sensor
- OmniMapper has a plugin for generic 3D iterative closest point (ICP)
- Finds overlap between sequential point clouds.
- ICP in OmniMapper is based on PCL

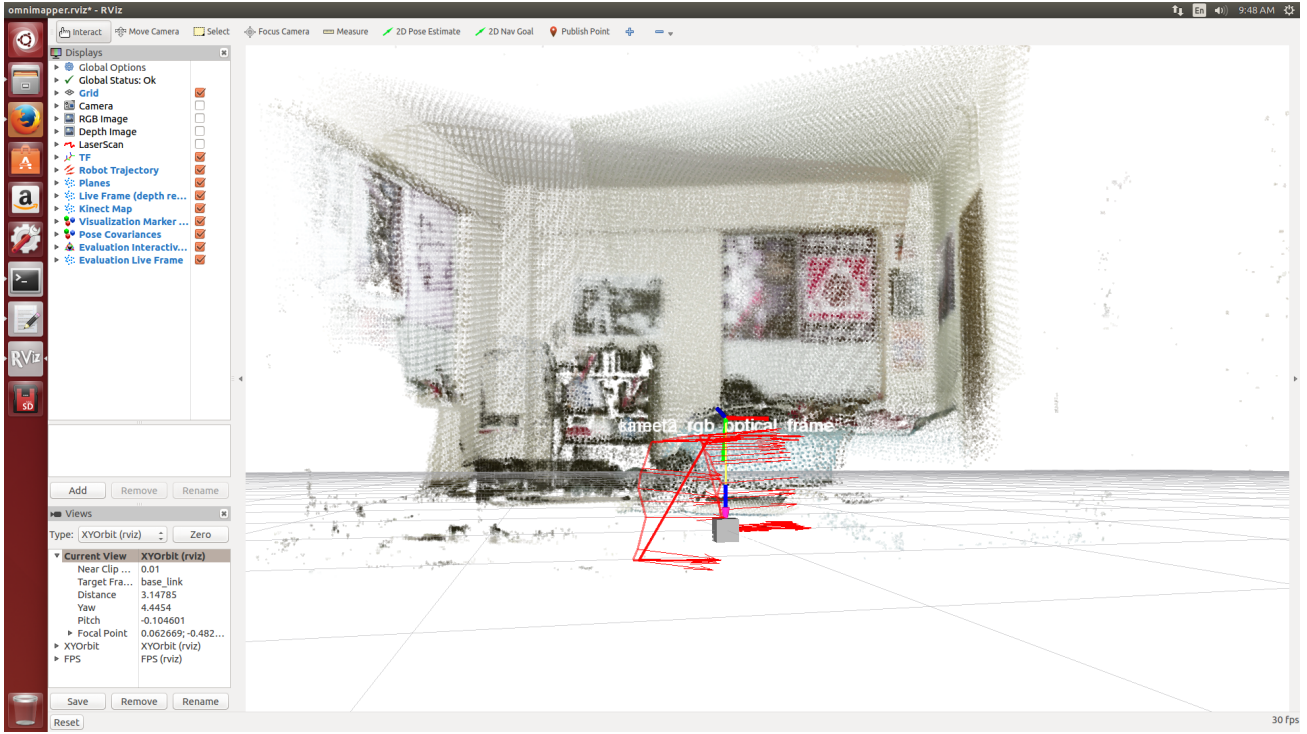


Kinect 2 Point Cloud

SLAM with Kinect 2

- Easily Integrated within ROS launch files+parameters
- Static transform publisher node for Kinect 2 frame

- Low Frequency (about ~1 Hz)
- Error for fast movement
  - Need high-frequency Odometry



Output of SLAM with just Kinect 2

SLAM with Odometry

Seeker Jr Robot Has Encoder-based Odometry built in (x,y,yaw)

Odometry added to SLAM with ROS tf

OmniMapper is also configured to use a transform from Odometry as an input to SLAM:

```
<param name="odom_frame_name" value="/odom"/>
 <param name="use_tf" value="true"/>
 <param name="tf_trans_noise" value="0.05"/>
 <param name="tf_roll_noise" value="10000"/>
 <param name="tf_pitch_noise" value="10000"/>
 <param name="tf_yaw_noise" value="0.2"/>
```

Straightforward code for sending tf

```
tfBroadcaster = new tf::TransformBroadcaster();
...
void odometry_update(){
double ex = robot->getEncoderX() / MM_IN_M;
double ey = robot->getEncoderY() / MM_IN_M;
double et = to_radians(robot->getEncoderTh());
    tf::Transform transform;
    transform.setOrigin( tf::Vector3(ex, ey, 0.0) );
    tf::Quaternion q;
    q.setRPY(0, 0, msg->et);
    transform.setRotation(q);
    tfBroadcaster.sendTransform(tf::StampedTransform(transform,
    ros::Time::now(), "odom",  "agribot"));
}
```

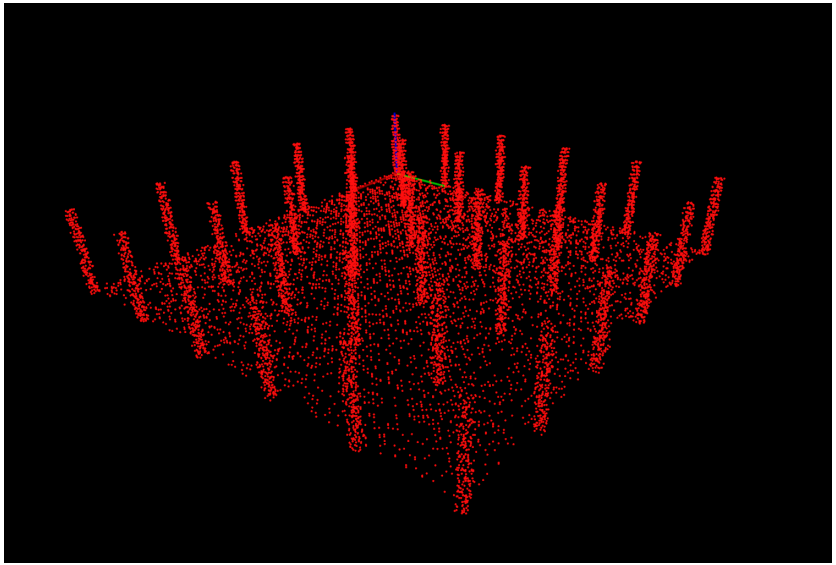
Plant Detection with PCL

- Need to somehow find plants within sets of 3D points
  - Simplifying assumption: plants are surrounded by empty space

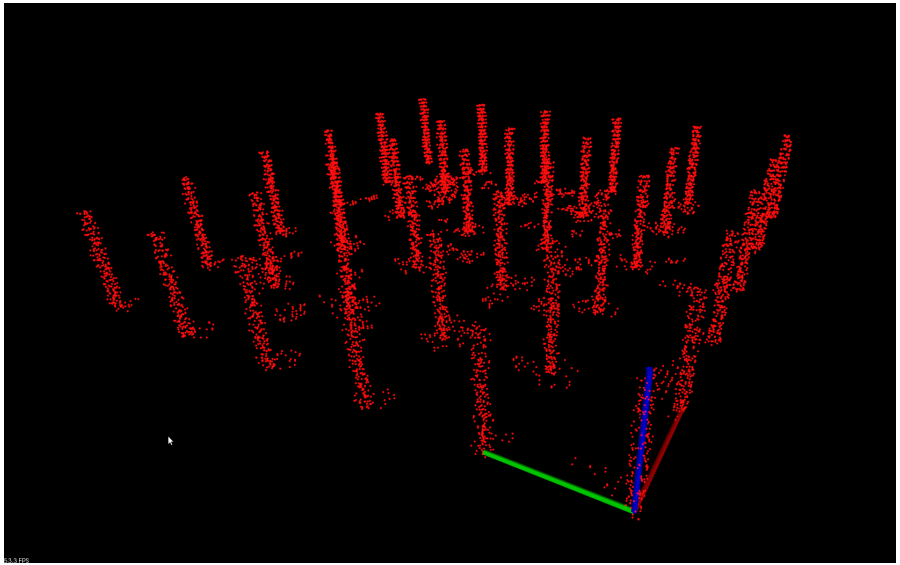
Use PCL to implement Euclidean Clustering+cloud filtering

1. Filter out noise by removing statistical outliers
2. Downsample to simplify cloud
3. Filter out points below some threshold (remove ground)
4. Build KDTree on this Point Cloud
5. Perform Euclidean Clustering to find plants

PCL Results in Synthetic point cloud

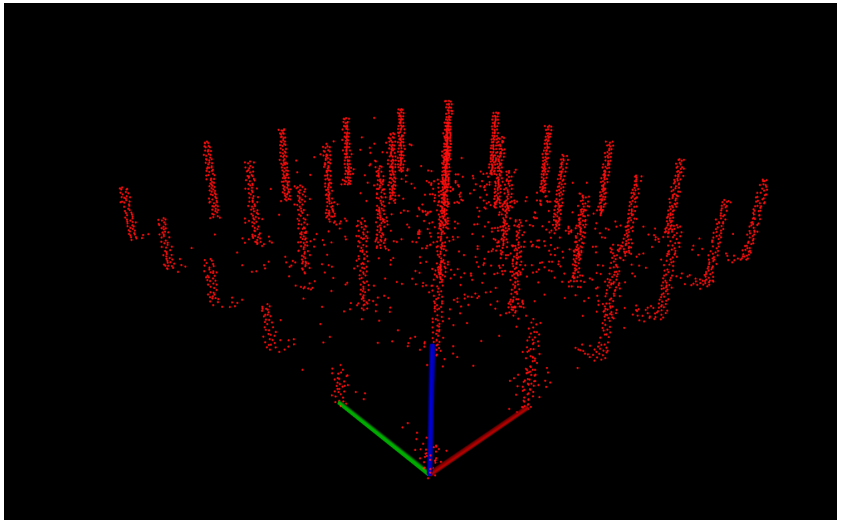


Initial Cloud

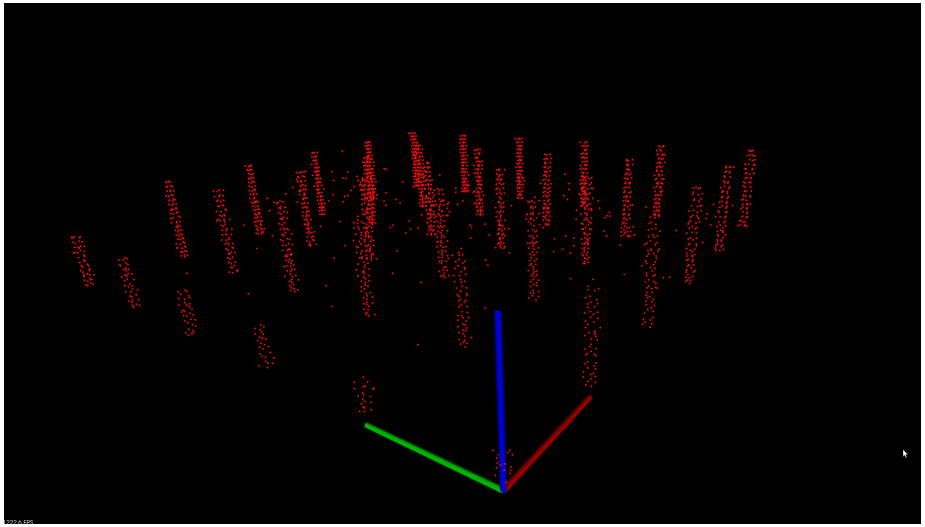


Statistical Outliers Removed

PCL Results in Synthetic point cloud

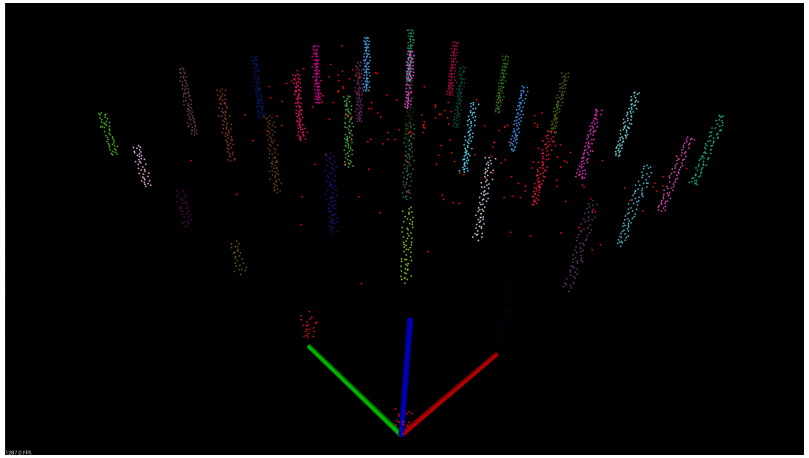


Downsampled Cloud

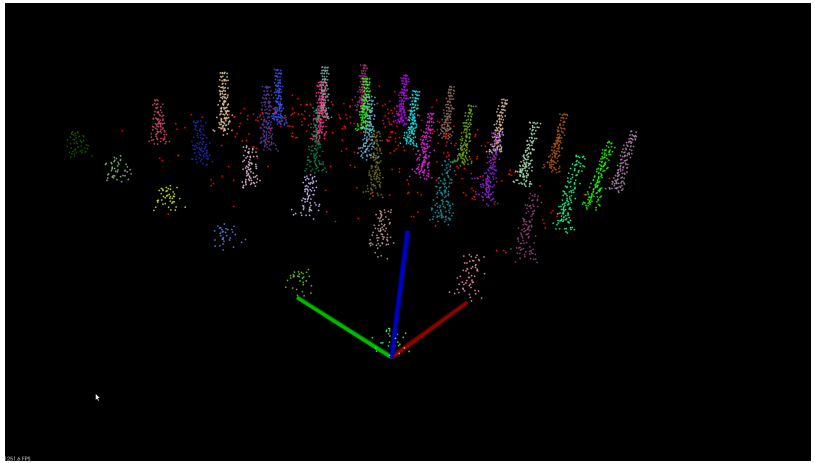


Height Thresholded Cloud

PCL Results in Synthetic point cloud



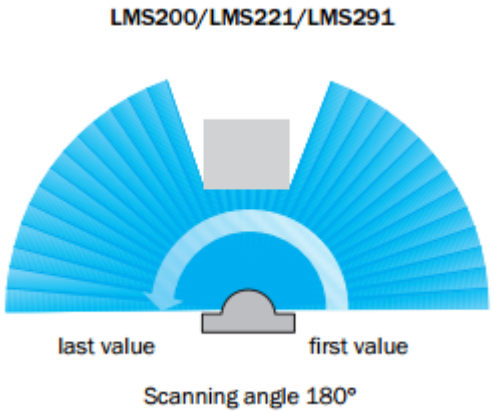
Found Clusters

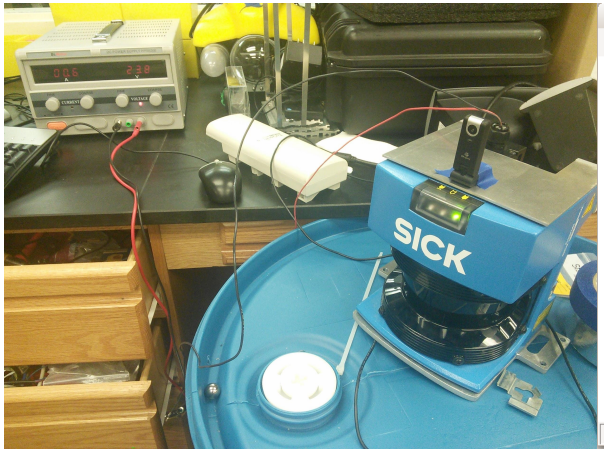


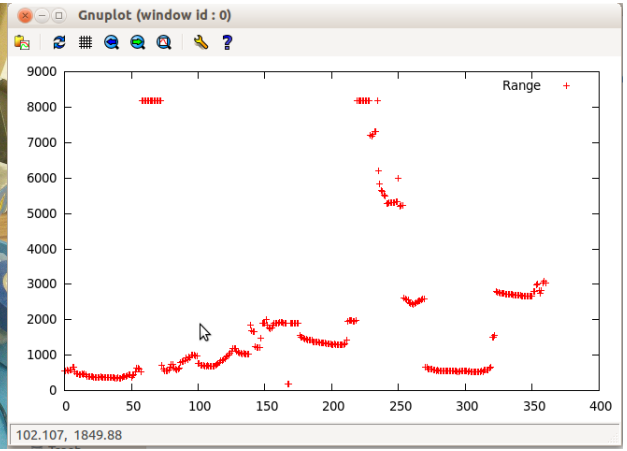
Found Clusters in noisier cloud

LIDAR

- SICK 200 (laser scan)
  - Made to work with sick toolbox
  - Allows us to detect obstacles







Full Integrated SLAM

LIDAR integrated as with other sensors

```
<node name="sick_node" pkg="sicktoolbox_wrapper" type="sicklms"
respawn="false" output="screen">
    <param name="port" value="/dev/ttyUSB0"/>
</node>
```

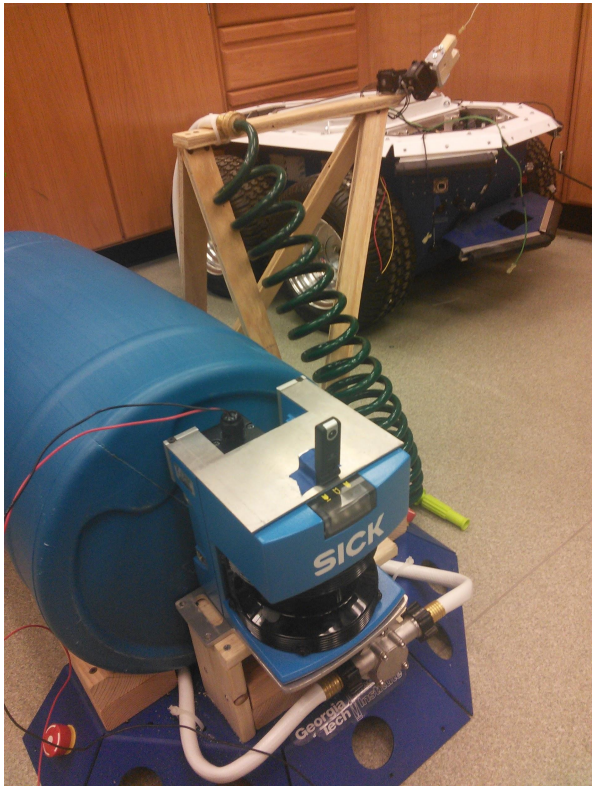
Then, in the omnimapper launch parameters, this is straighforwardly added:

```
<param name="use_csm" value="true"/>
```

As can be seen in the OmniMapper source code, this makes the package subscribe to the output of the the created sicklms node (which outputs to the topic "scan"):

```
if (use_csm_)
 {
   // Subscribe to laser scan
   laserScan_sub_ = n_.subscribe ("/scan", 1,
    &OmniMapperROS::laserScanCallback, this);
```

Due to hardware problems on the robot, not yet tested



LIDAR Obstacle Detection

Implemented simple distance-based approach to LIDAR obstacle detection

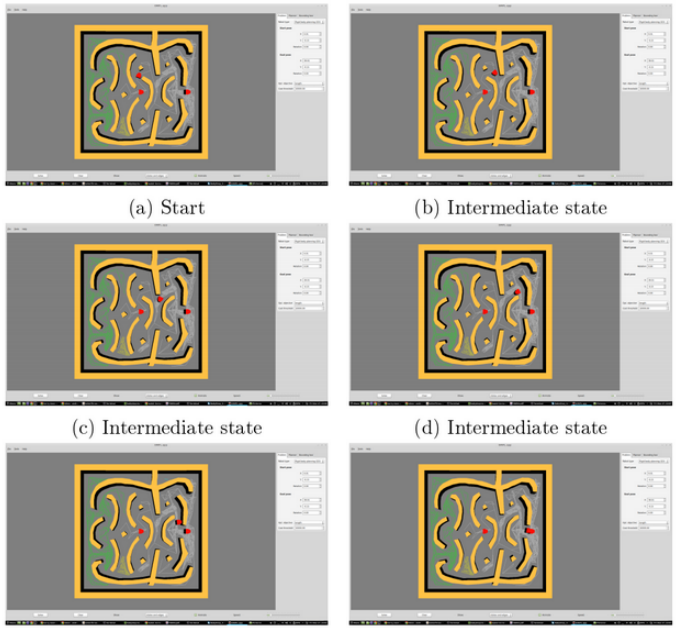
```
void lidarCallback(const sensor_msgs::LaserScan::ConstPtr& msg){
    obstacleInFront = false;
    for(int i=0; i <msg->ranges.size();i++){
        float dist = msg->ranges[i];
        float angle = msg->min_angle + i*(msg->angle_increment);
        if(angle>-20 && angle<20){
            if(range<OBSTACLE_DIST_THRESHOLD){
                obstacleInFront = true;
                break;
            }
        }
    }
}
```

Navigation

- Based on data from SLAM, the robot makes navigation decisions
- Which plant to spray next, how to get there

Path Planning with OMPL

- OMPL (Open Motion Planning Library)
- 20-30 planners
- Various state spaces supported, e.g., SE(n), kinematic car model, R^n

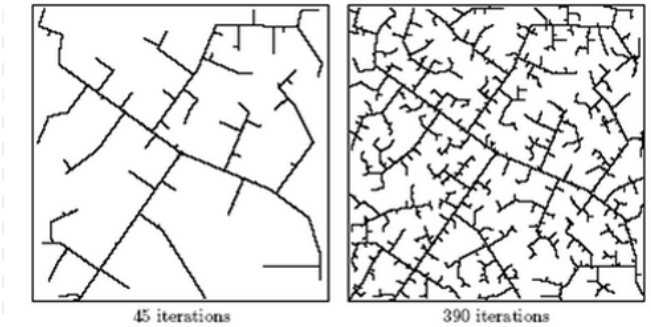


(a) Start (b) Intermediate state (c) Intermediate state (d) Intermediate state (e) Intermediate state (f) End.

Figure 5: Results of using PRM planner with SE(2) object.

OMPL Planners: RRT

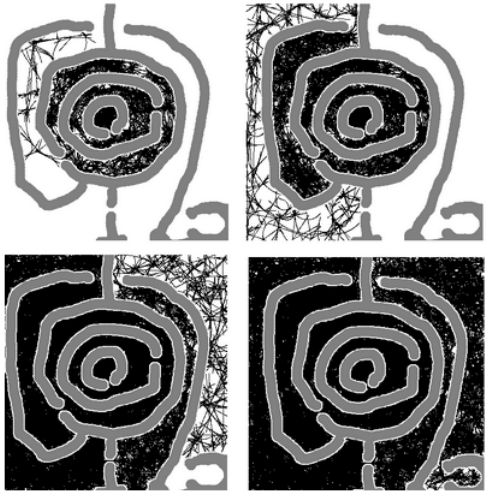
- Introduces the concept of a tree-based planner
  - Starts from the initial state and randomly walks outward, making sure not to collide with obstacles
- Can improved by simultaneously growing two trees, one from the initial state and one from the goal state



Source: S. M. LaValle's *Planning Algorithms*, p. 230

OMPL Planners: PDST

- Another tree-based planner
- A score is assigned to each cell of the state space based on its volume and a □□priority□□ measure
- When moving from sample to sample in the search, the □□ next sample is defined as the one with the lowest score

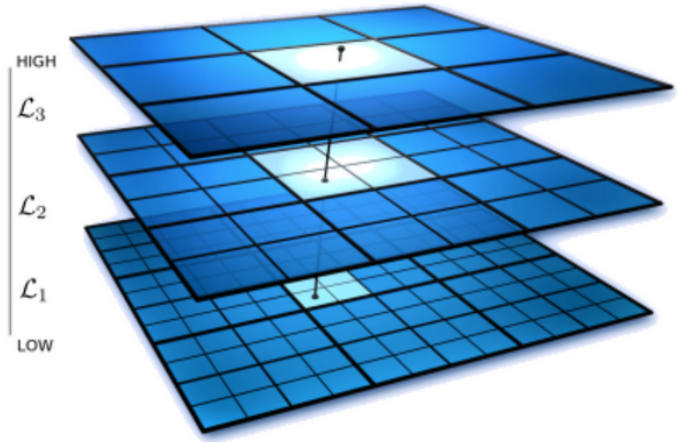


Exploring reachable free space of a second-order differential drive robot using PDST

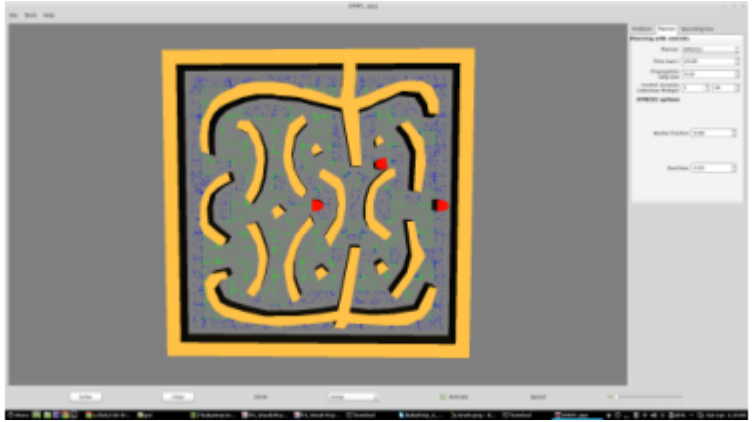
RICE Kavraki Lab

OMPL Planners: KPIECE

- Tree-based planner
- Takes the state space and projects it into a grid
- There are multiple levels of grid, each lower level constructed by chopping up the grid at the previous level
- The figure from the authors of KPIECE illustrates the multiple levels of discretization



OMPL Planners: Results on Dummy Map



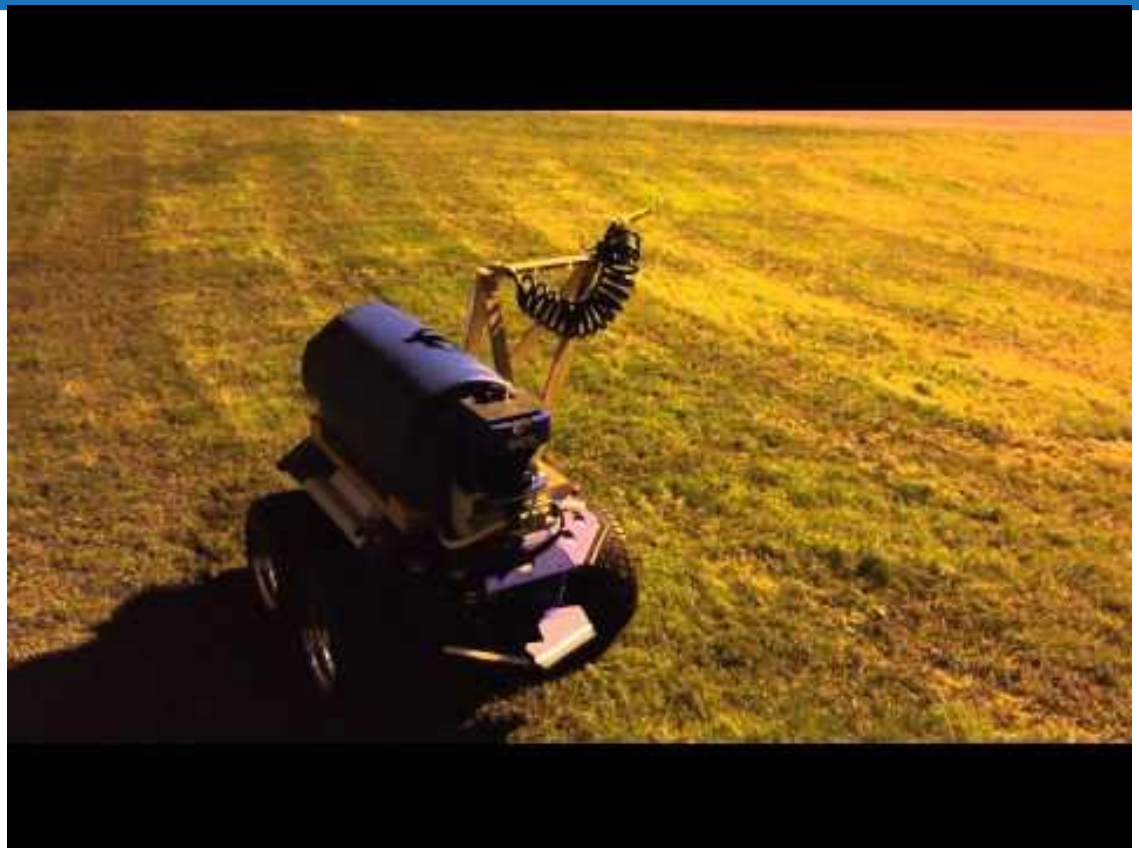
Platter Type	Planning Time (Min)	Path Time (20S planning)
KPIECE	7s	65s
EST	5s	51s
PDST	6s	44s
RRT	1s	63s

Table 1: Constraint robustness results by skill type.

OMPL Integration Status

- Basic problems with the robot, such as the
  - odometry resetting incorrectly
  - other low-level issues such as frayed wires
- Focus is on repairing low-level basic functionality before testing full functionality
- OMPL to be integrated after low level problems fixed

Current Status Video



Next Steps

- Resolve all low-level issues with robot
- Integrate OMPL into current navigation code for robust path-planning
- Test integrated SLAM on robot
- Test plant detection on data from Kinect 2

Robot in the Wild

Questions?