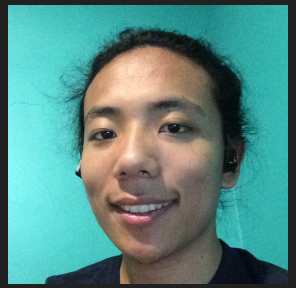
Team 5: Tokyo Drift

Lab 5A-B: Mapping, Localization, and Path Planning

Eric Wieser

Ernie Ho



Shi-Ke Xue

Steven Homberg

Winter Guerra

Goals

Mapping:

- Collect Tf, Odom, LaserScan data
- Convert result into map

Localization

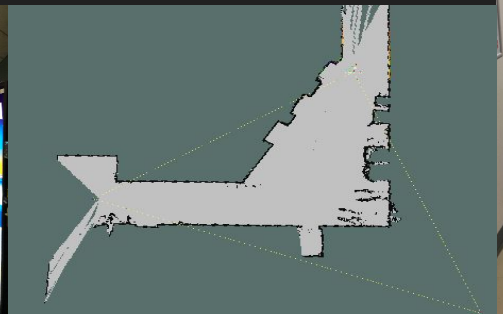
- Determine position in map
- Update with odometry

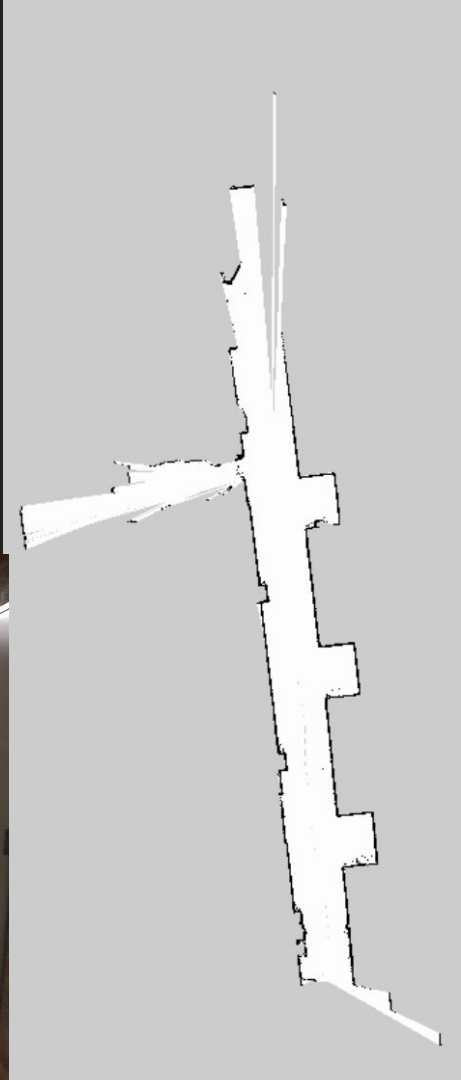
Control:

- Follow predefined path through map

Data Collection and Mapping

- gmapping, alright ...
- hector-slam, better
- First signs of time issues - need 1/20 speed bag playback to allow time for updates
- Odom, base_link and map misplacement

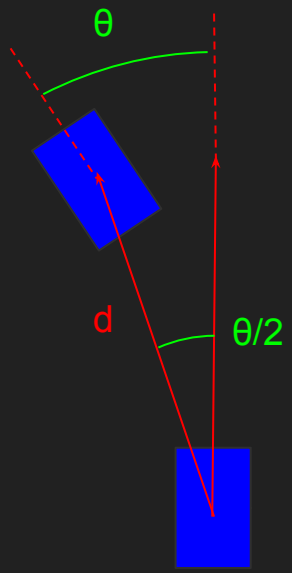




Localization in Four Parts

- Draw initial particles
- Motion update moves particles
- Sensor update assigns weights
- Draw new particles

Motion Updates



$\theta = \omega * dt$

$d = v_x * dt$

Integrate (approximately)

- Forward velocity
- angular velocity

Assume no lateral drift

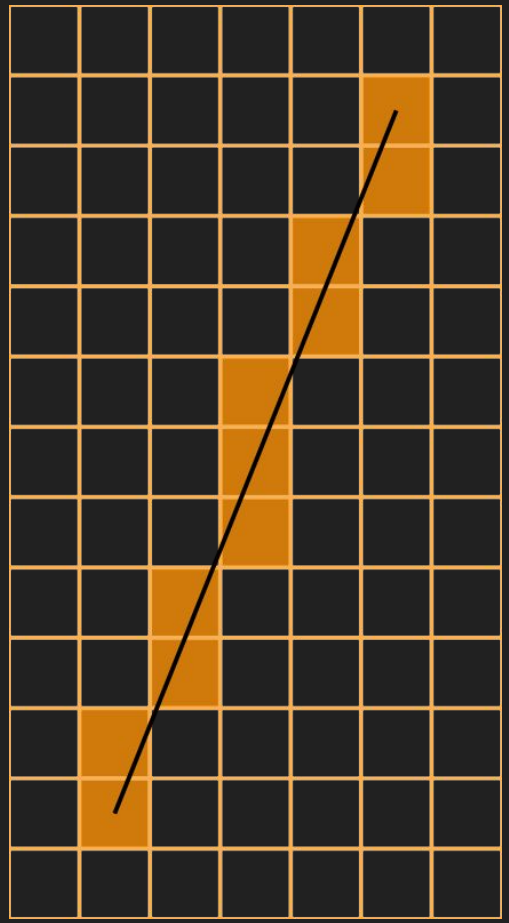
Modelling uncertainty in motion updates

- Assume that the steering angle and wheel speed are distributed normally
  - $\sigma_{angle} = 10°$, $\sigma_{speed} = 0.2ms^{-1}$
- Propagate variance and covariance through equations for dynamics
  - Not all functions of gaussians are gaussian - use taylor approximations

- Populate the missing `odometry.twist.covariance` matrix in `/vesc/odom`
- Sample from the resulting distribution using `np.random.multivariate_normal`, and integrate the samples

Sensor Update

- Calculate expected ranges (ray casting)
- Find error from readings
- Return weight of particle



Making Sensor updates run (almost) fast enough

- Throw away 90% of the laser data (960 → 96 ranges)
- Vectorize the python using numpy

  This:

```
for p in particles:
    for angle in angles:
        x = p.x + math.sin(angle)
        y = p.y + math.cos(angle)
```

  Becomes

```
dxs = particles[...,np.newaxis].x + np.sin(angles)
dys = particles[...,np.newaxis].y + np.cos(angles)
```

- Write a native module for python:
  - ~~Using Cython~~ not supported officially by catkin
  - ~~In C, using the CPython API~~ steep learning curve
  - In plain C, with python executing it via `ctypes`

Comparison of performances

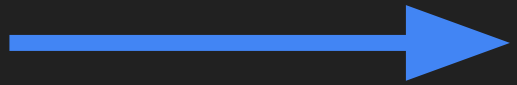
Tested on 20 particle with 100 laser scans

	Seconds to completion
First test	10
With numpy vectorization	3.8
With C code	2.4
With numpy and C code	0.13

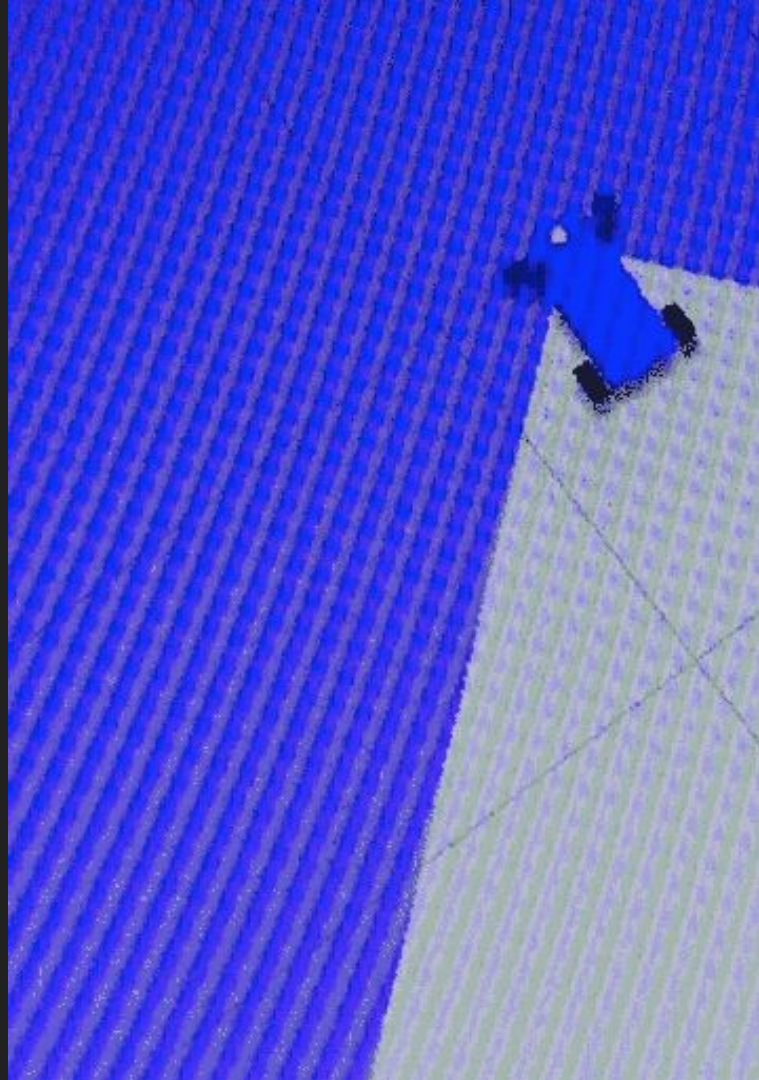
To get odometry in Gazebo we created a new node that pulls from Gazebo internals

Why did we do this?

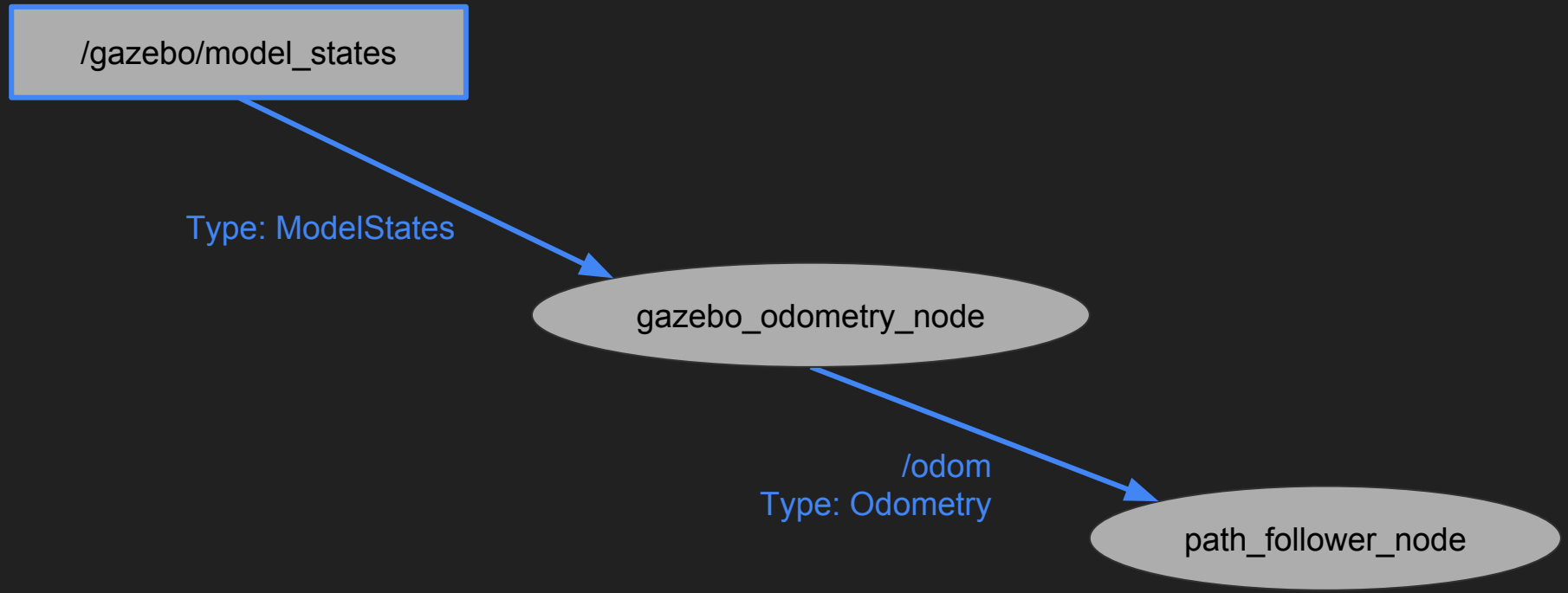
Because the stock odometry patch for Gazebo causes this to happen



(Hint: Look at the front wheels)

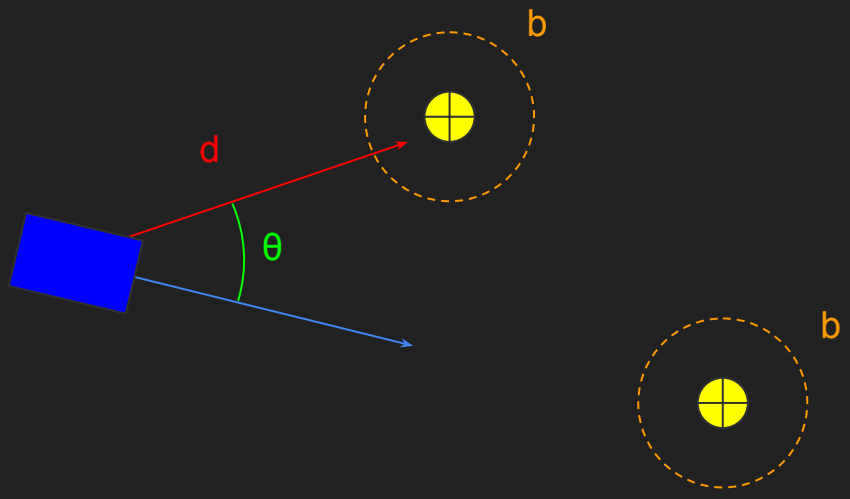


To fix this problem, we created an odometry node that publishes odometry from Gazebo internals



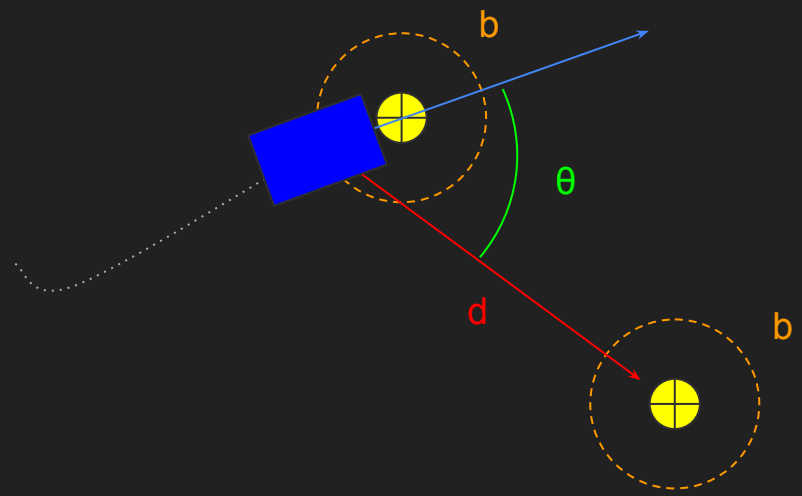
Now, onto path following

To follow a path, the robot follows waypoints using odometry data and PID



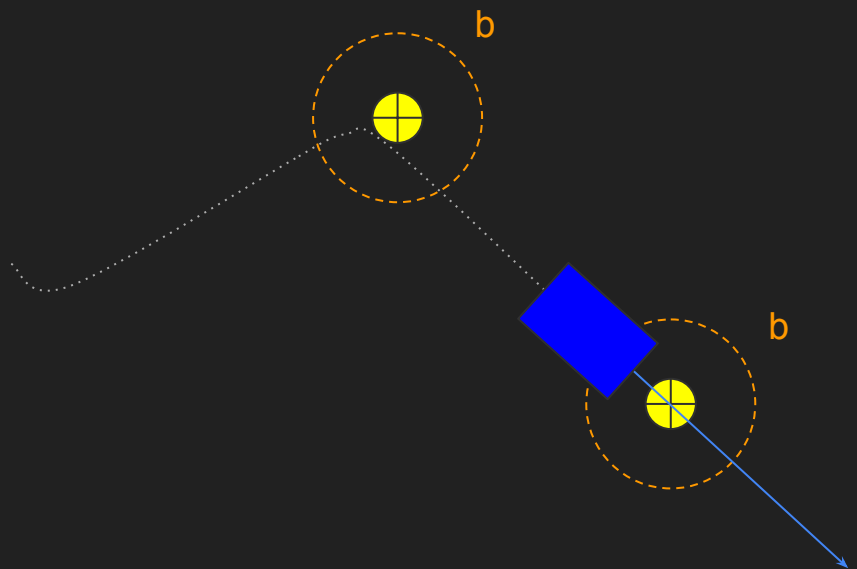
d = distance error
θ = angle error
b = waypoint boundary

To follow a path, the robot follows waypoints using odometry data and PID



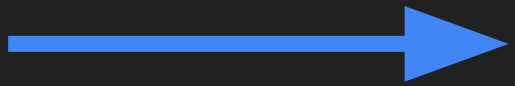
d = distance error
θ = angle error
b = waypoint boundary

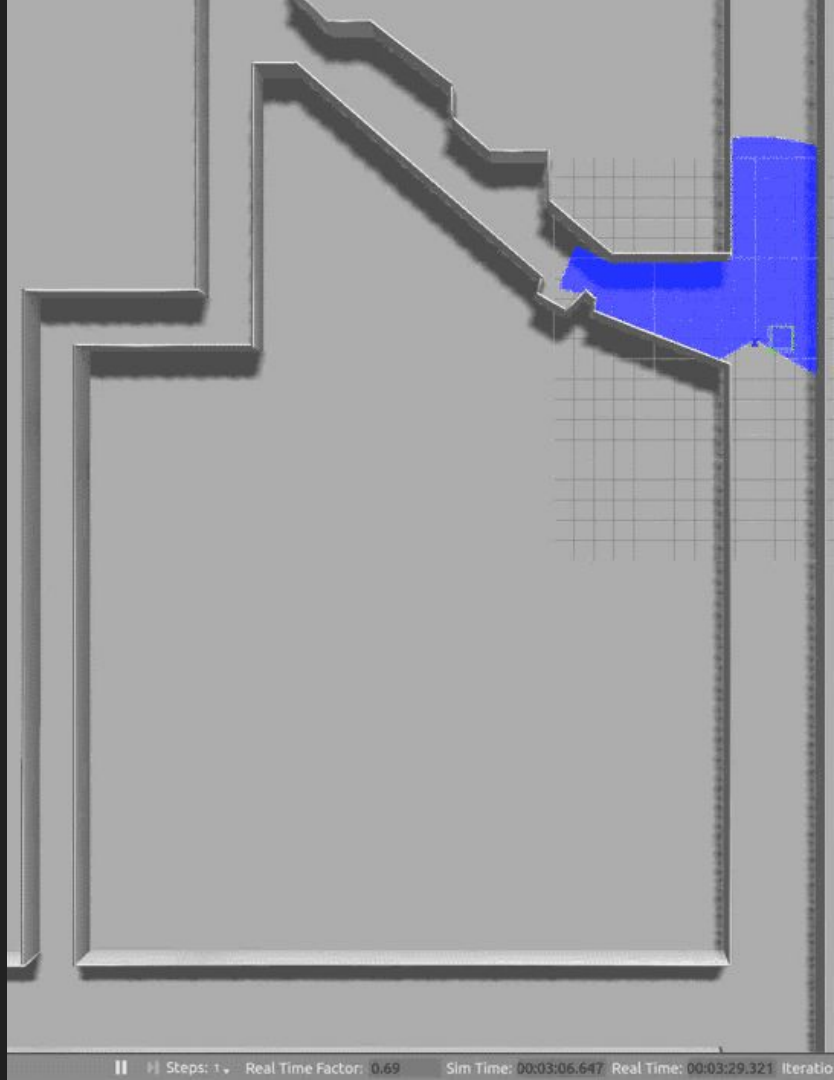
To follow a path, the robot follows waypoints using odometry data and PID



d = distance error
θ = angle error
b = waypoint boundary

Here’s the result of our path following algorithm





Results

Mapping:

- Obtained reasonable, if noisy maps of several test environments in usable format

Localization:

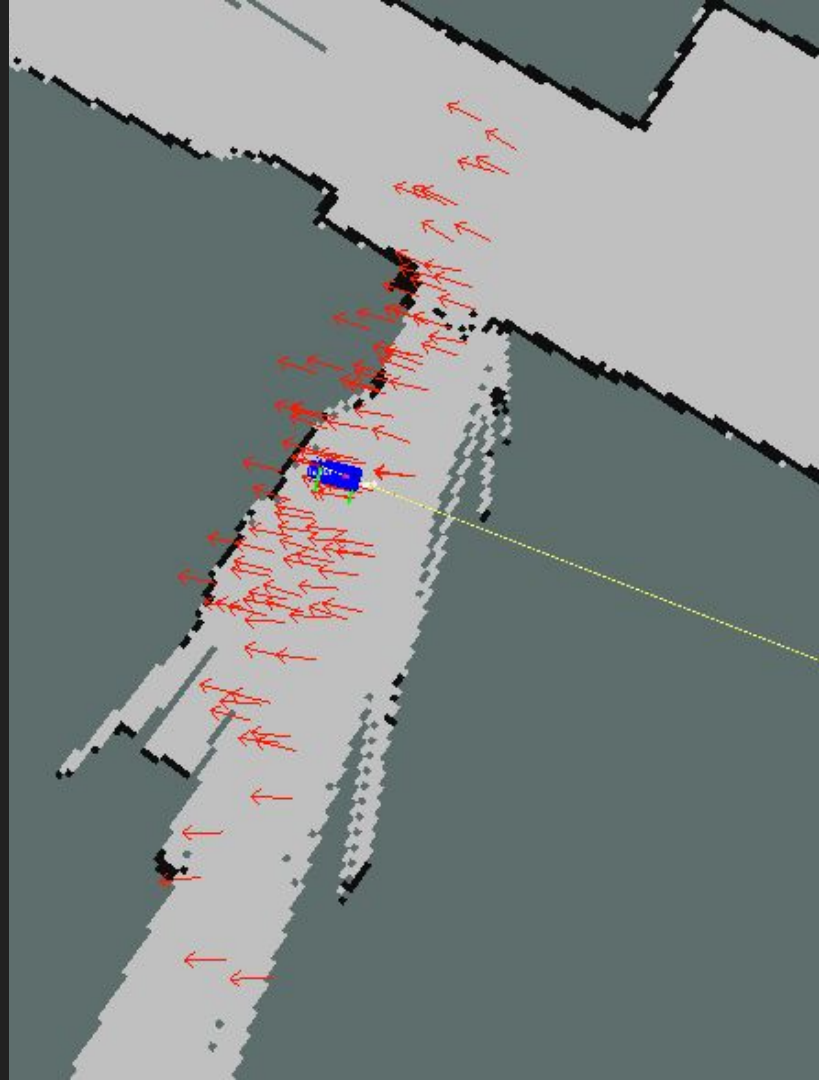
- Developed semi-functional particle filter localization
  - Too slow to run real-time
  - Sometimes gets lost

Control:

- Successfully followed predefined path (in simulation)

Lessons Learned

- Python is slow
- Python is less slow when vectorized using numpy
- Python is even less slow when you replace the slow bits with C
- Slow algorithms are slow whatever you express them in
- Merge upstream code into a *separate branch* to test so that breaking upstream changes do not break code on master.



Contributions

Eric: Wrote unit tests, and launch files for mapping visualization. Added abstraction layer over `OccupancyGrid`. Vectorized particle filter code with numpy.

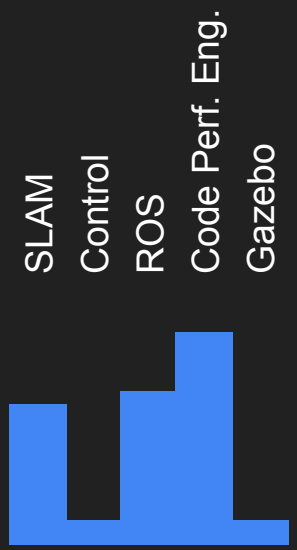
Ernie: Map odom base_link misplacement, data collecting and hector mapping.

Steven: Implemented top-level filtering, motion updates; integrated faster map ranging with ctypes.

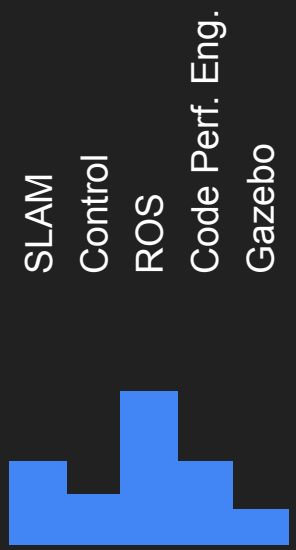
Winter: Created odom node. Debugged odom issues with Gazebo. Wrote PID *path_follower* node.

Shi-Ke: Worked on particle filter, primarily the sensor update step.

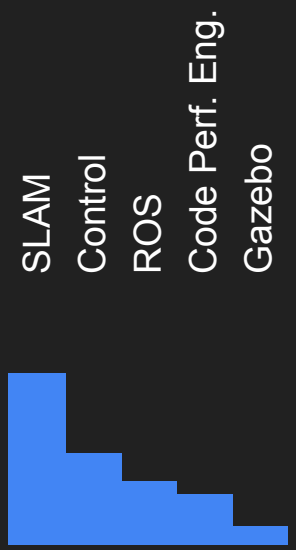
Individual Learning



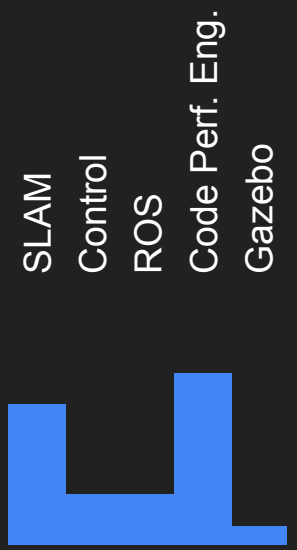
Eric Wieser



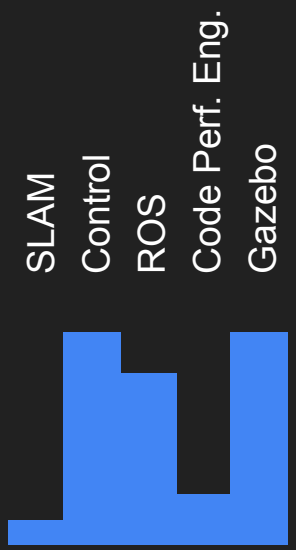
Ernie Ho



Shi-Ke Xue



Steven Homberg



Winter Guerra

Graphs show normalized amount learned