Final Mission - Lego Mindstorms EV3 Localization

Hardware

- Lego Mindstorms EV3 Brick (300Mhz ARM9, 64MB RAM)
- 2 Large Motors
- 2 Ultrasonic Sensors
- 2 Color Sensors
- Edimax EW–7811Un 150Mbps 11n Wi-Fi USB Adapter

Software

- ev3dev environment based on Debian Linux (3.16 kernel)
- python-ev3 Python development environment

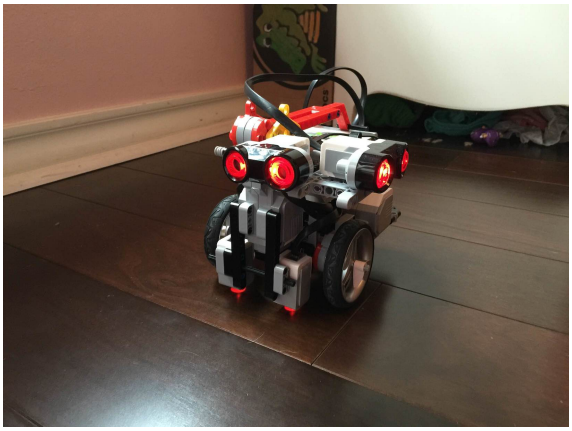
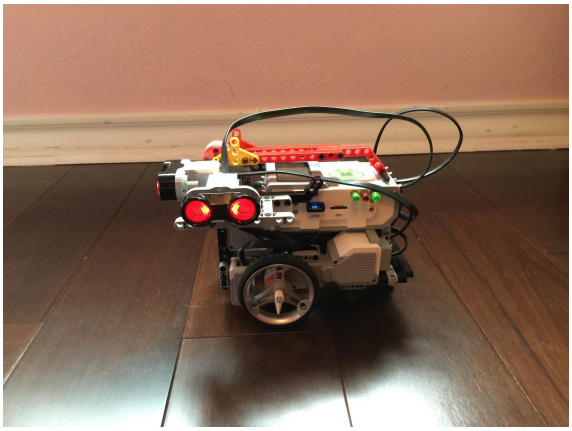
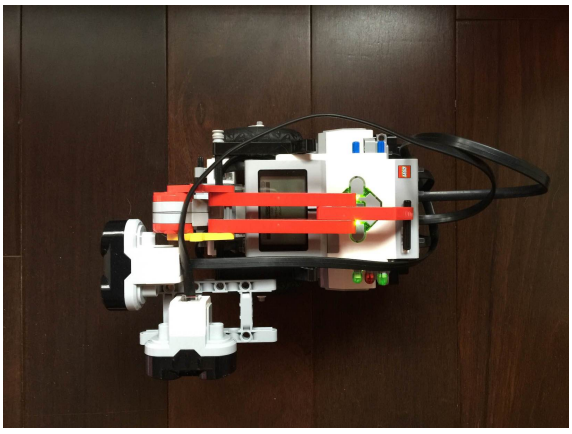
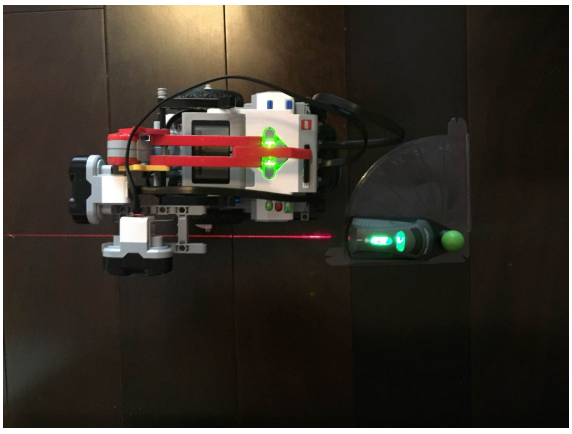
Software Installation

1. Follow installation instructions for ev3dev, using ev3dev-jessie–2014–10–07 image
2. Follow installation instructions for Python EV3

Robot Design

The robot is a fairly compact differential drive system with 2 large motors powering the front wheels, and a caster ball in the rear. The color sensors are optimally positioned ahead of the wheels (allowing for quick wheel response to sensor readings), about 0.5cm off the ground. The color sensors are only used to locate the target after successful global localization. The ultrasonic sensors are also positioned ahead of the wheels. One ultrasonic sensor faces left (pi radians) and is used for wall following. The other faces forward (pi/2 radians) and is used to detect

walls or obstacles ahead.

Program Overview

The program uses ultrasonic sensors to follow the left wall of a room. It uses a particle filter to perform global localization with a known map (it does not know its starting position/orientation). Once enough steps (manually configured) have been taken for successful localization, it calculates a heading and distance towards a target (white business card on the floor) and moves towards it using simple dead reckoning. At the

estimated target location, it uses a spiral motion and 2 color sensors to locate the target. Note that the particle filter is run after every step, but this could also be done after several (or all) steps as the robot motion is based on the wall following routine.

Program Details

For program code, please look at **ev3_wall_trace_localize.py** or click here to access the file online.

Program Flow

Initialization:

1. Initialize occupancy grid and list of walls
2. Initialize particle filter

Sense/Update/Move (Loop):

1. Sense (ultrasonic)
2. Particle filter measurement update
3. Particle filter re-sampling
4. Move (wall follower)
5. Particle filter motion update

Target acquisition:

1. Move towards target based on localization estimate
2. Search (spiral move) and sense (color)

Wall Follower

The wall follower uses a PD controller to maintain a constant distance from the left wall with the left facing ultrasonic sensor. The wall follower works reasonably well with just a P controller, but the PD controller limits overshoot and stabilizes the oscillation. The PD controller calculates the appropriate speed ratios for the left/right motors. The front facing ultrasonic sensor detects walls or obstacles ahead, and triggers hard right turns.

Motion

As a standard differential drive robot, motion is controlled by the speed ratios of the left/right motors. The robot moves straight when both motors run at the same speed/direction. It turns in place when both motors run at the same speed, but in opposite directions. Other ratios result in the robot moving in a circular arc.

For real robot motion, the motors are run with “regulation mode” enabled to ensure that they actually maintain the requested speed. With “regulation mode” disabled, the difference in speed from one motor to another results in unpredictable behavior. In addition, motors are always set to “brake at end” instead of “coast”.

For the simulated particle motion, gaussian noise is applied both to the position, as well as the orientation. The position noise applied is a gaussian with a standard deviation of 5cm (the typical robot step is about 18cm). The orientation noise applied is a gaussian with a standard

deviation of 0.03 rad (1.7 deg). Turning in place results in a higher orientation noise. This is modeled as a gaussian with a standard deviation of 0.05 rad (2.9 deg). The motion noise values ensure a healthy particle population.

Sensing

The ultrasonic sensors are challenging to work with as they can sometimes return rubbish values. Testing shows that the sensors need to be within a +/- 25 deg angle to the wall to operate, and accuracy decreases with distance. The maximum range is 255cm, but above 200cm, readings are unreliable. As such, the maximum distance for ultrasonic measurements is fixed at 200cm. All ultrasonic readings are repeated 3 times and the lowest value is selected.

For the simulated particles, the distance to the closest wall (in the list of walls) is calculated. This involves several checks:

- the angle between the sensor and the wall must be within the sonar cone
- the forward distance from the sensor to the wall must be less than the maximum sonar distance (200cm)
- the intercept point on the wall must be within the ends of the wall

If no valid walls are found, a distance of 200cm is returned. This minimizes the measurement error at higher distances.

The sonar noise, used in calculating the measurement probability, is a gaussian with a standard deviation of 10cm for the left sensor (the wall follower routine aims to maintain a 25cm gap), and 15cm for the front sensor. Note that the front sensor is typically measuring over much longer distances so the accuracy decreases. These higher sensor noise values ensure that the good particles are not inadvertently killed off.

Localization

The particle filter is initialized with 500 particles in random positions (orientations randomly chosen out of N, S, W, E). At 500 particles, the EV3 takes about 10s at every step to perform particle filter computations. Running with 1000 particles does not appear to warrant the additional computational load. During particle filter initialization, the occupancy grid is checked to ensure that particles are not placed in an occupied cell.

The measurement probability is a gaussian error calculated on the difference between the measured sonar distances, and the calculated distances to the closest walls, given the specified sonar noise for each sensor. A constant (0.000001) is added to make for a more robust likelihood function. This also limits the extent to which good particles are killed off as a result of spurious bad readings. The occupancy grid is also checked during measurement probability calculation, and any particles that are out-of-bounds or in an unavailable cell have their probability reduced to zero.

During re-sampling, particles are randomly sampled according to their

measurement probabilities (a given particle may be sampled multiple times). As the robot takes more unique corners, the particles converge around the actual robot location. The estimated position is calculated by averaging the positions of the converged particle set.

Running the Program

Running on Real Robot - Single Particle Mode (Data Gathering/Debugging)

This mode gathers measurement data from the real robot for use with subsequent offline simulations. It also provides additional debugging output.

```
run_steps = 25 (set to desired number of steps)
single_particle_mode = True
real_robot_mode = True
grab_target = False
```

To add a new data set, save the “measurements” output at the end of the run, and include the number of steps, start position, and end position (see existing data sets for more information).

Running on Real Robot - Full Particle Filter

This mode, used for the final robot run, runs the full particle filter on the real robot at every step, performs global localization (starting position/orientation unknown), and attempts to grab the target after the specified steps have completed.

```
run_steps = 25 (set to desired number of steps)
single_particle_mode = False
real_robot_mode = True
grab_target = True
```

Running on Simulator - Single Particle Mode (Debugging)

This mode simulates single particle mode offline with a saved data set. It initializes the single particle to the known starting position, and provides additional debugging. It also compares the final estimated position with the known ending position.

```
single_particle_mode = True
active_set = 1 (specify saved data set)
real_robot_mode = False
```

Running on Simulator - Full Particle Filter (DEFAULT)

This mode simulates full particle filter mode offline with a saved data set. It runs the full particle filter at every step, performs global localization (starting position/orientation unknown) and compares the final estimated position with the known ending position. This is the primary mode used for all optimization and testing.

```
single_particle_mode = False
active_set = 1 (specify saved data set)
real_robot_mode = False
```

Results - Real Robot Runs

Program Output

To view the program output for a real robot run (single particle mode) used to generate data set #1, please look at **real_run_single_particle_stop_1.txt** or click here to access the file online.

To view the program output for a real robot run (single particle mode) used to generate data set #2, please look at **real_run_single_particle_stop_2.txt** or click here to access the file online.

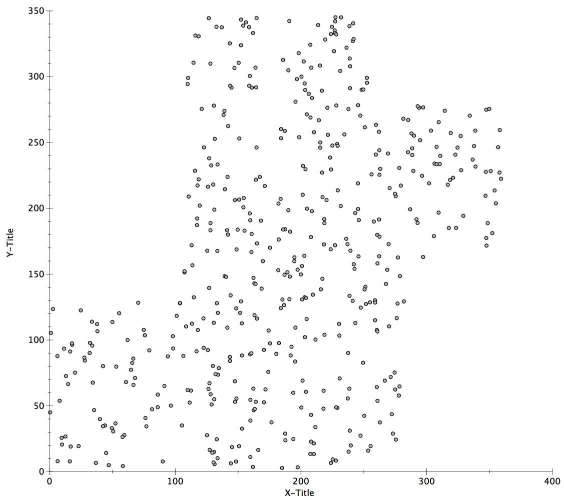
To view the program output for the final run performing full particle filtering on the EV3, please look at **real_robot_final_run.txt** or click here to access the file online.

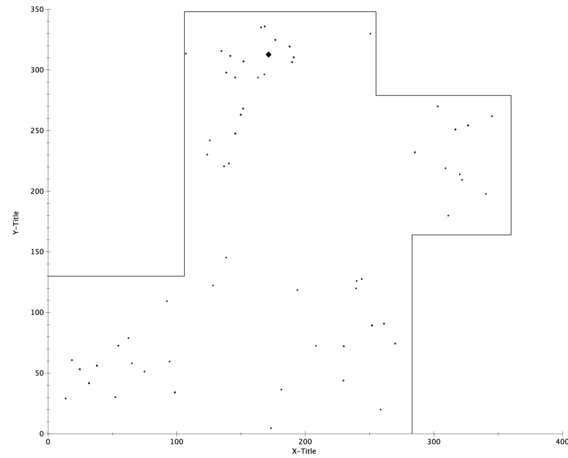
Videos

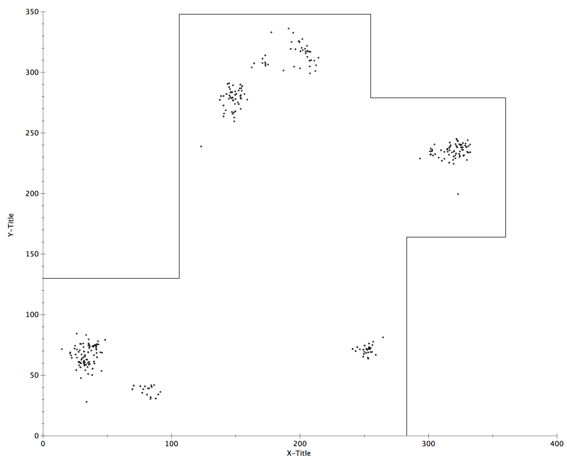
To view a video of a real robot test run (single particle mode - known starting position), please look at **test_run_single_particle_known_start.MOV** or click here to access the video online. Note that this is very similar to the final run video, but without the 10s stop at every step to perform particle filtering.

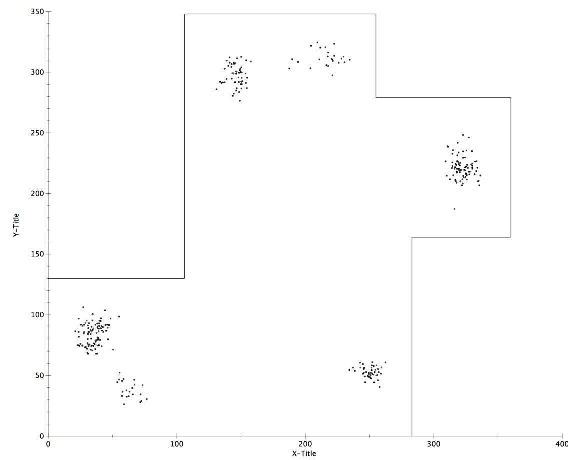
To view a video of the final run performing full particle filtering on the EV3, please look at **final_run.MOV** or click here to access the video online.

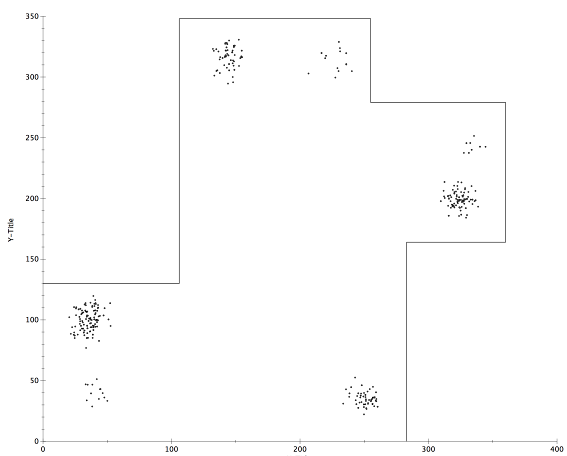
Particle Maps

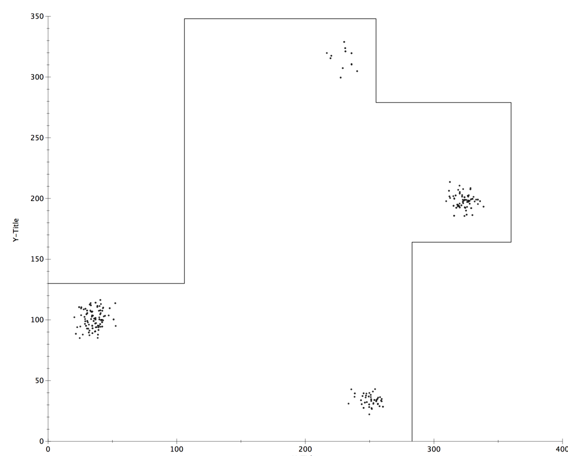

Y-Title
X-Title
0
100
200
300
400
50
150
250
350

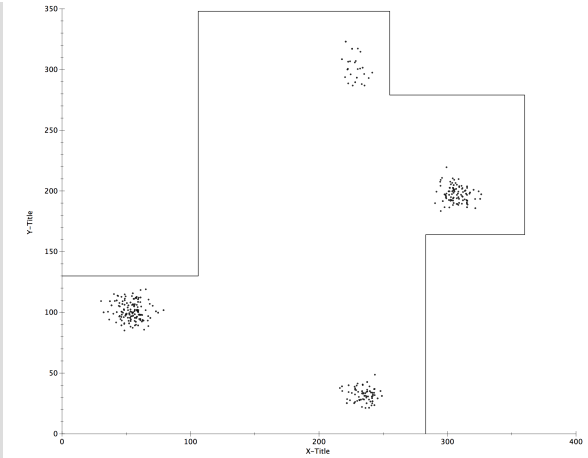

Y-Title
X-Title

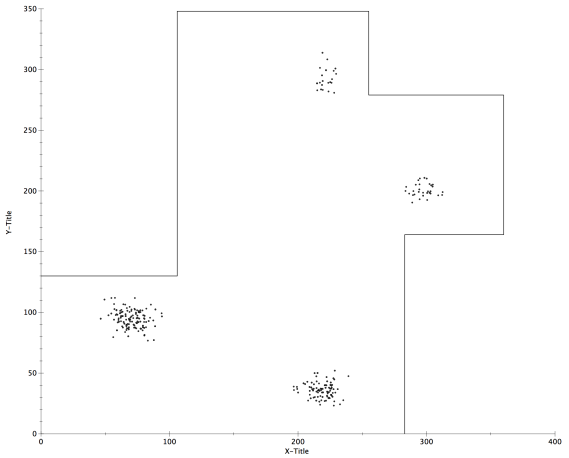

Y-Title
X-Title

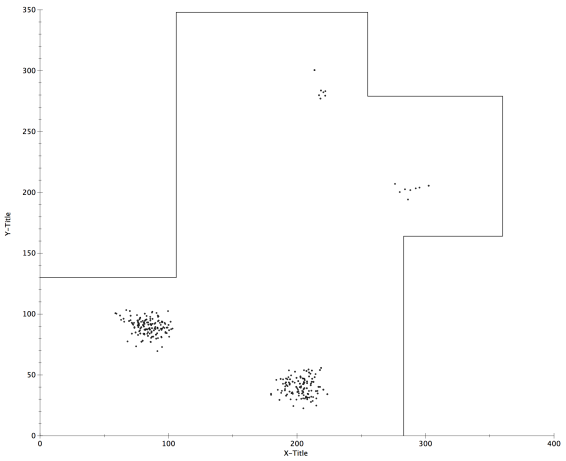

Y-Title
X-Title

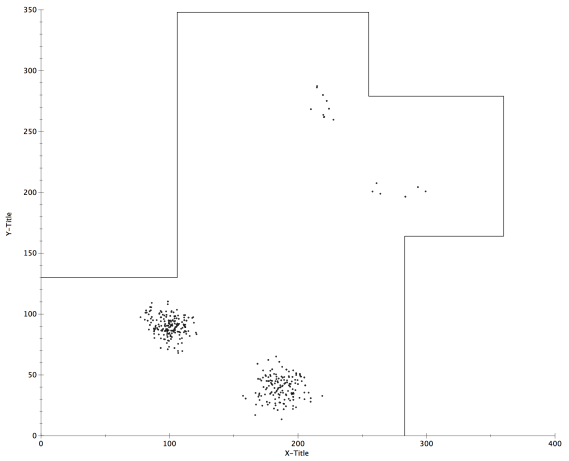

Y-Title
X-Title

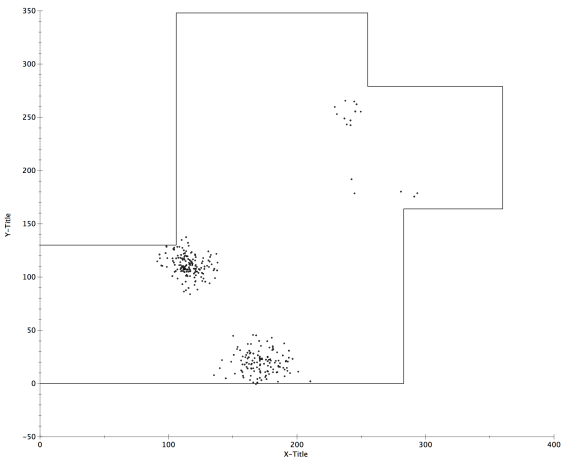

Y-Title
X-Title

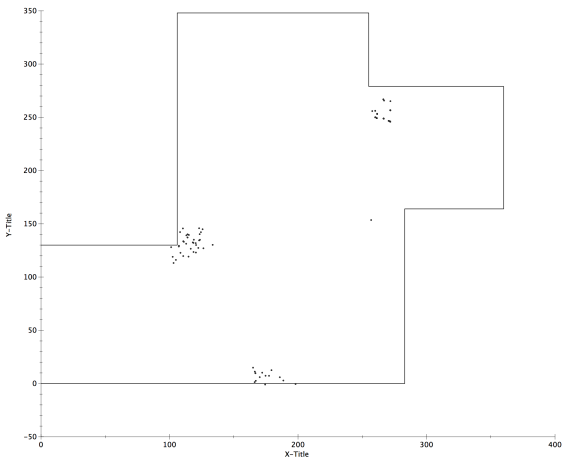

Y-Title
X-Title
0
50
100
150
200
250
300
350
0
100
200
300
400

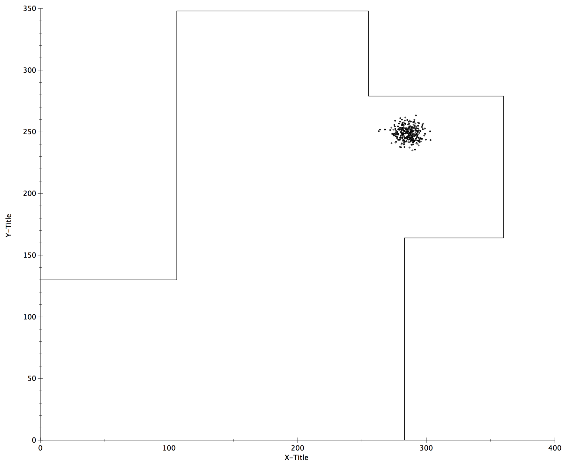

Y-Title
X-Title
0
50
100
150
200
250
300
350
0
100
200
300
400

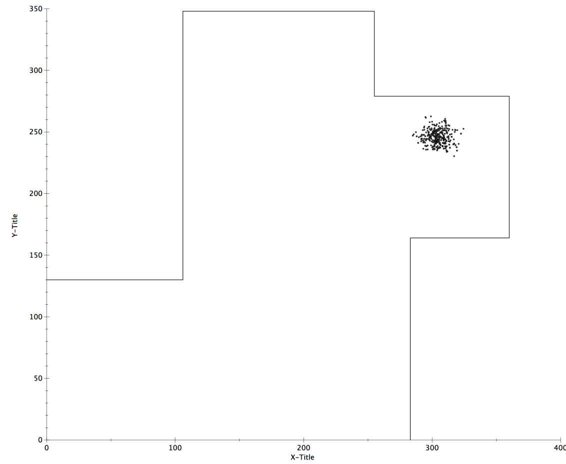

Y-Title
X-Title
0
50
100
150
200
250
300
350
0
100
200
300
400

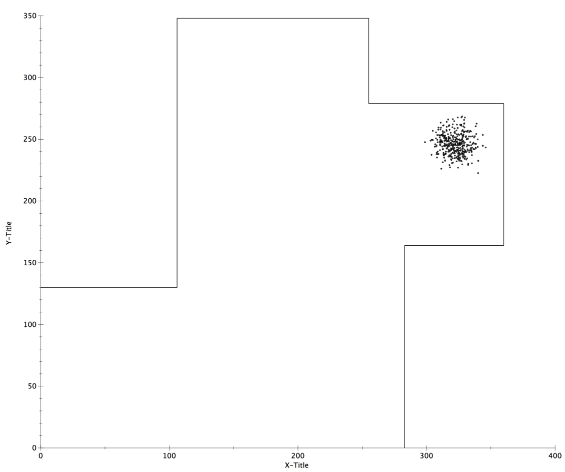

Y-Title
X-Title
0
50
100
150
200
250
300
350
0
100
200
300
400

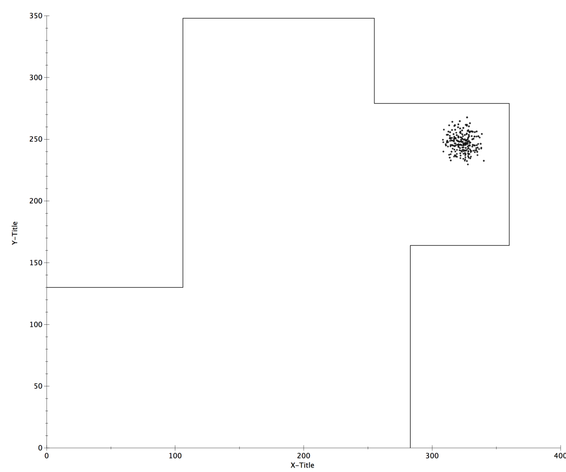

Y-Title
X-Title
-50
0
50
100
150
200
250
300
350
0
100
200
300
400

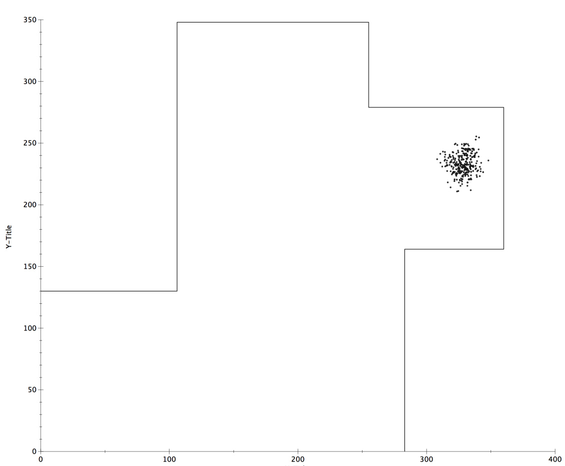

Y-Title
X-Title
-50
0
50
100
150
200
250
300
350
0
100
200
300
400

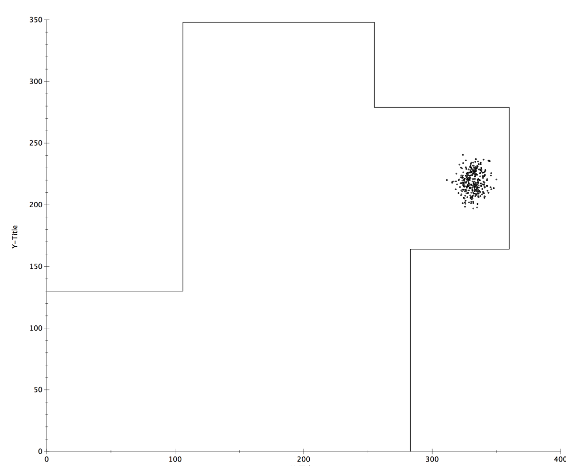

Y-Title
X-Title

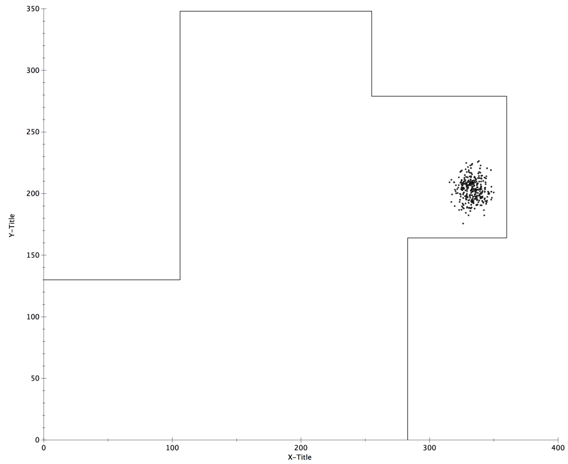
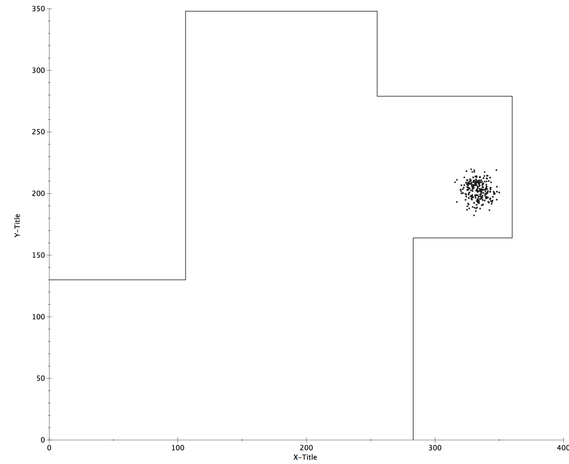
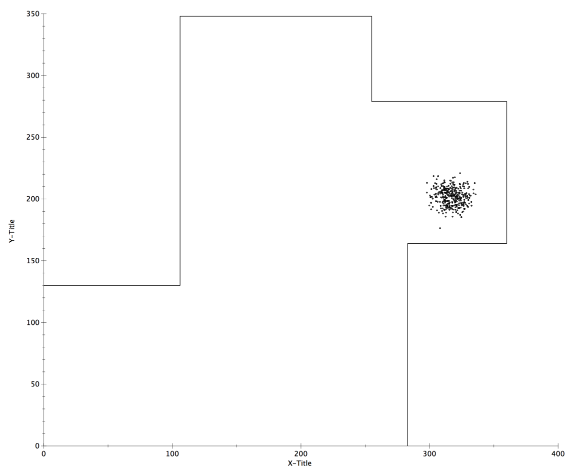
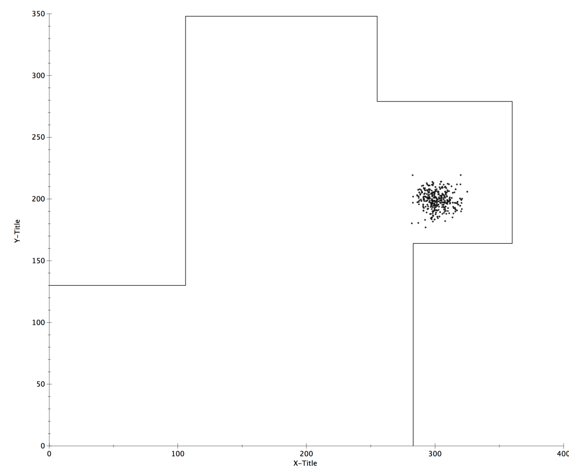
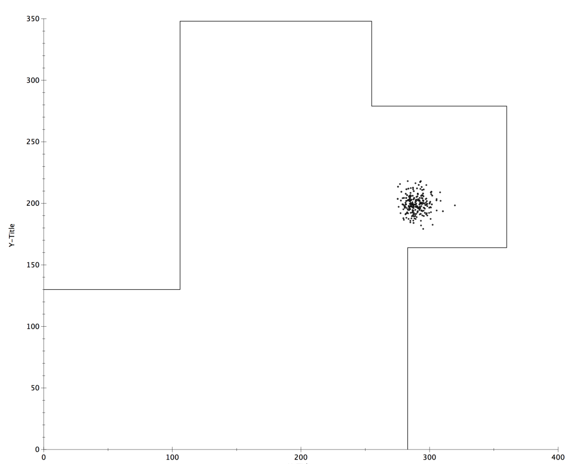
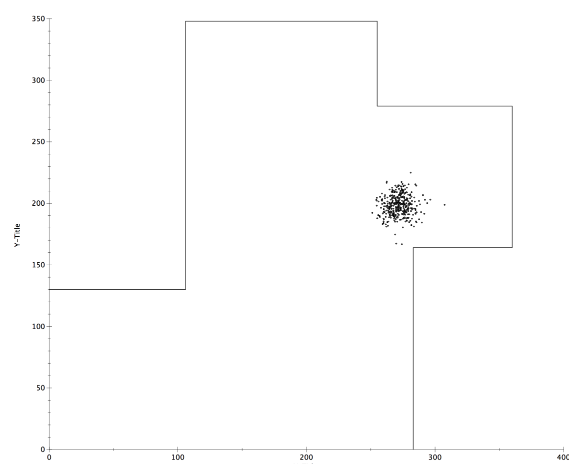

Y-Title
X-Title

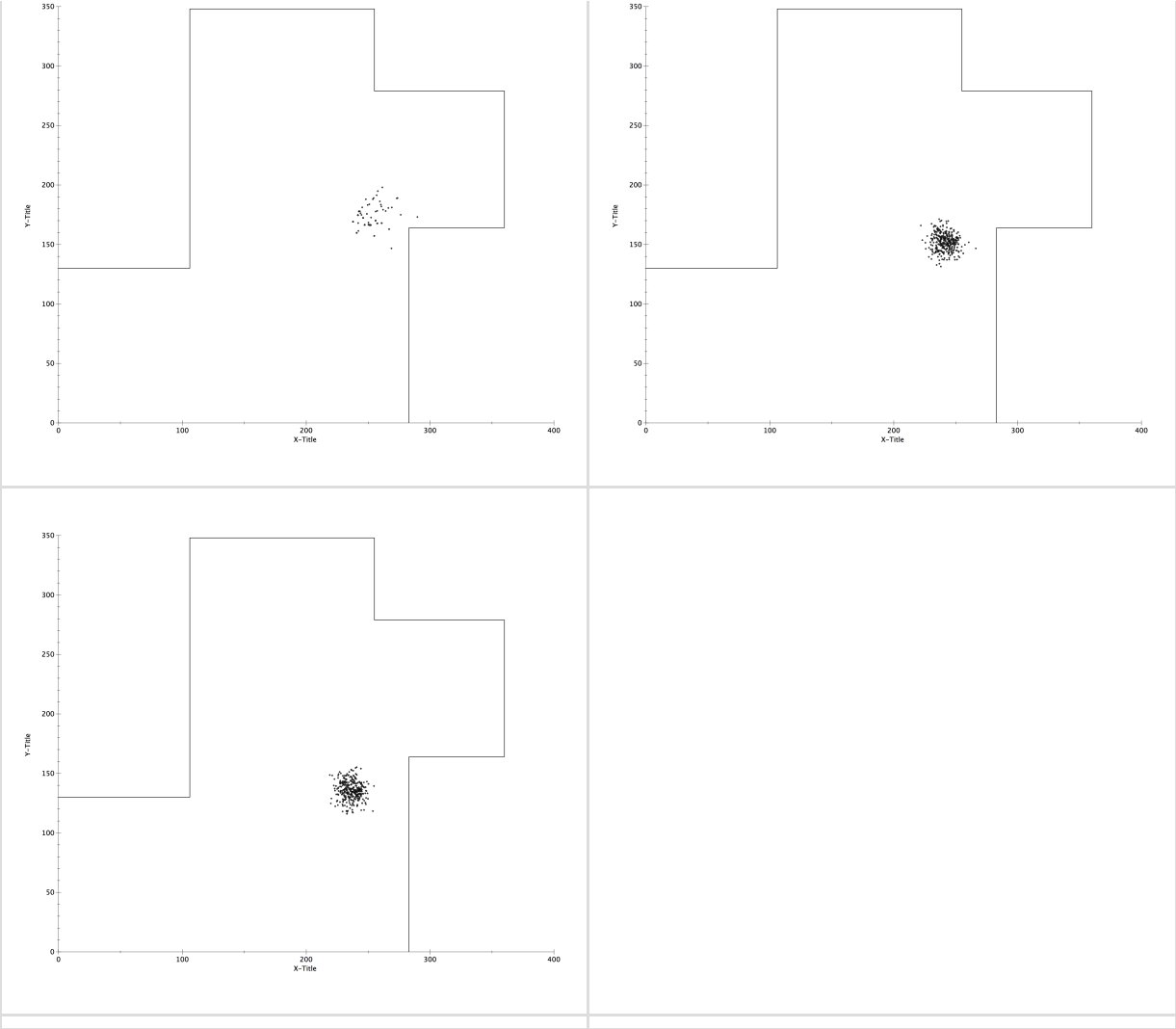

Y-Title
X-Title


Y-Title
X-Title


Y-Title
X-Title


Y-Title
X-Title


Y-Title
X-Title
0
50
100
150
200
250
300
350
400

Y-Title
X-Title
0
50
100
150
200
250
300
350
400

Y-Title
X-Title
0
50
100
150
200
250
300
350
400

Y-Title
X-Title
0
50
100
150
200
250
300
350
400

Y-Title
X-Title
0
50
100
150
200
250
300
350
400

Y-Title
X-Title
0
50
100
150
200
250
300
350
400



Results - Simulation Runs

Observations

Estimated positions generally align very well with real measured positions. A consecutive set of runs is shown below. This is based on the default program configuration (a simulation run using a full particle filter against data set #1). Occasionally, there is an incorrect result. While it may be possible to minimize this through additional particle filter tuning, it would be more beneficial to provide the robot with additional sensor input. For instance, right or rear facing ultrasonic sensors, or spinning the robot in

place at every step. The addition of a compass sensor would provide a simple improvement. This can be partially simulated by setting **known_starting_orientation** to **True** in the program.

Run #1

```
Final estimated position:  [241.58867759090208,
119.50400720232773, 4.999448212215737]

Final real position:  [251.0, 111.0, 5.11]

Final position deltas:  [9.41132240909792, 8.504007202327728,
0.11055178778426367]
```

Run #2

```
Final estimated position:  [244.9753832611407,
112.36134788014402, 4.984630565170516]

Final real position:  [251.0, 111.0, 5.11]

Final position deltas:  [6.024616738859294, 1.3613478801440237,
0.12536943482948448]
```

Run #3

```
Final estimated position:  [242.94027020248646,
100.55441386472299, 5.024042259670237]

Final real position:  [251.0, 111.0, 5.11]

Final position deltas:  [8.05972979751354, 10.445586135277011,
0.08595774032976333]
```

Run #4

```
Final estimated position:  [247.5128161974207,
113.07498297656565, 4.964009458599826]

Final real position:  [251.0, 111.0, 5.11]

Final position deltas:  [3.4871838025792954, 2.07498297656565,
0.14599054140017476]
```

Run #5

```
Final estimated position:  [245.76305795382356,
119.90726560810684, 4.888702177348583]

Final real position:  [251.0, 111.0, 5.11]

Final position deltas:  [5.236942046176438, 8.90726560810684,
0.22129782265141706]
```

Program Output

To view the program output for a simulation robot run (single particle

mode) using data set #1, please look at **simulation_run_single_particle.txt** or click here to access the file online.

To view the program output for a simulation robot run (full particle filter) using data set #1, please look at **simulation_run_full_pf_1.txt** or click here to access the file online.

To view the program output for a simulation robot run (full particle filter) using data set #2, please look at **simulation_run_full_pf_2.txt** or click here to access the file online.

Assumptions/Limitations

- The robot always starts alongside a wall.
- The collision avoidance during wall following is limited to the basic wall following algorithm. There is no collision avoidance when looking for the target.

References

- A PID Controller For Lego Mindstorms Robots, http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html
- Lecture 2: Robot Motion, [http://www.doc.ic.ac.uk/ajd/Robotics/RoboticsResources/lecture2.pdf] (http://www.doc.ic.ac.uk/ajd/Robotics/RoboticsResources/lecture2.pdf)
- Lecture 5: Monte Carlo Localisation,

[http://www.doc.ic.ac.uk/ajd/Robotics/RoboticsResources/lecture5.pdf]
(http://www.doc.ic.ac.uk/ajd/Robotics/RoboticsResources/lecture5.pdf)

- ev3dev, http://www.ev3dev.org/
- python-ev3, https://github.com/topikachu/python-ev3