Implementing Particle Filter Localization on a Mobile Robot with *ROS*

Emre AY
eay@kth.se

ABSTRACT In this project, particle filter based localization system of a custom made mobile robot is developed. The robot was a differential-drive robot with an equipped laser sensor. Robot Operating System (ROS) framework is used to accomplish the project and write the source code. The practical aspects of implementing particle filter on a real hardware is reported.

KEYWORDS Estimation, Localization, Particle Filter, Robot Operating System, Mobile Robot

Introduction

Localization is a vital sub-system in all mobile robotics applications and it is desired to have a reliable localization system in order to accomplish tasks by integrating several sub-systems such as localization, planning, navigation and so on. Reliably estimating the position of the robot is the first step to make integration work in a robot system. Hence mobile robotics and localization are very popular application areas for estimation.

The robot in this report was a custom built differential-drive robot[†] as shown in Figure 1 which equipped with two DC motors with encoders, an RPLIDAR laser scanner, an Intel Realsense RGBD camera, a robotic arm, an Intel NUC mini computer and other necessary hardware such as drivers, power circuits, battery etc. The dimensions of the robot were 24×25 *cm* in depth and width and 17 *cm* in height from ground to the third layer. The robot was built for a search and rescue task on a maze, where localization is a key sub-system.

The selection of the filter was made between Extended Kalman Filter (EKF) and Particle Filter (PF). PF is selected to be implemented because;

- the first order Taylor expansion in linearization for EKF is open to errors for the regions away from the linearization region, yet nonlinearities are easier to handle with PF,
- PF is non-parametric, hence any distribution can be represented with particles such as multi modal/skewed distributions,
- due to its nature, global localization is very straightforward with PF
- multiple hypothesis tracking is trivial with PF

However, the computational cost is significantly higher in PF than in EKF. In order to succeed in the tasks, robot's pose is needed to be calculated in high enough rate. Hence reaching rate requirements was the main concern during the implementation of PF.

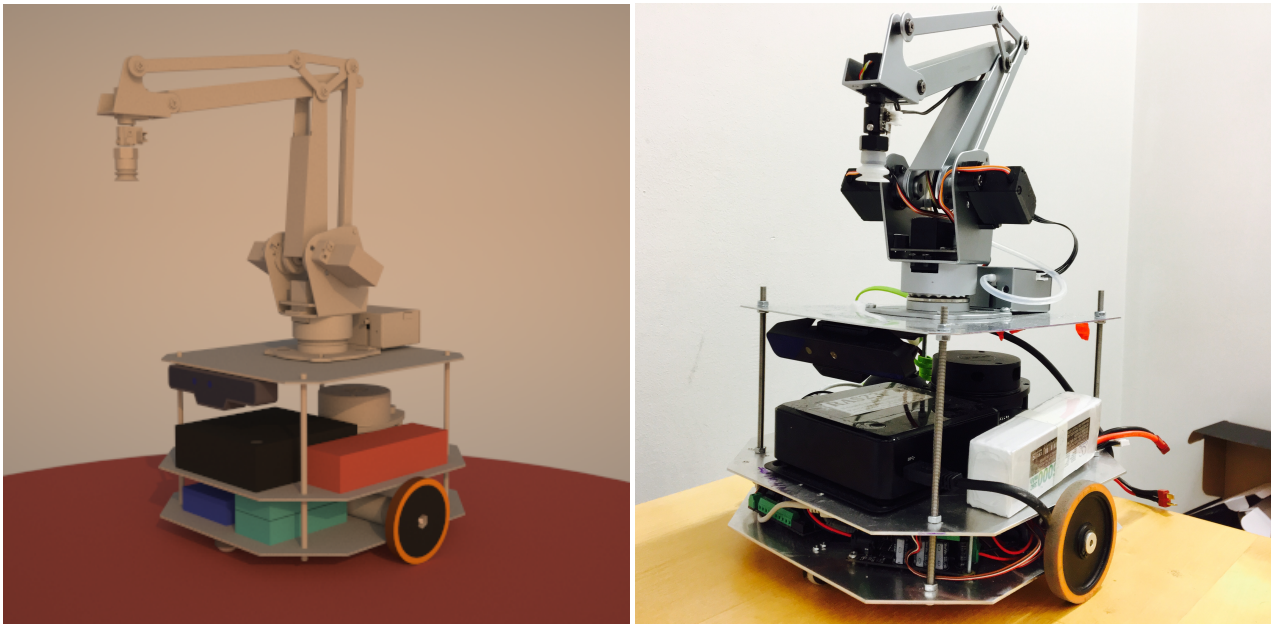
Robot Operating System (ROS) framework has shown an increasing popularity at robotics applications especially in academy since it was first released at 2009 by Willow Garage. The hardware on the present-day robots differ broadly. To prevent writing codes again for same or similar tasks on robots with different hardware, or in others words to avoid reinventing the steel by providing an environment is the basic logic behind ROS. ROS is designed to be "peer-to-peer, tool based, multi-lingual, thin, free and open-source"[1], hence ROS serves as a middle-ware and provides basic communication protocols together with debug features to be used in our programs called *nodes*. There are many official and unofficial open-source packages for ROS such as sensor/device drivers, message types etc. There are also packages directly for PF localization (i.e. amcl package), however in this implementation no external packages were used for particle filter localization. Only `boost/random`, standard ROS and C++ libraries are included.

In order to use the odometry information in the *predict* phase of the PF and also to compare the results, a dead reckoning system was implemented as a node. Dead reckoning node was simply reading from left and right motor encoders and calculating the odometry information of the robot based on its kinematics. It was also *publishing* necessary information on a ROS *topic* to be used in the PF predict.

The sensor to be used in the PF *update* phase was RPLIDAR A1 laser scanner. It is a relatively cheap laser scanner with 360° measurements per scan, 6 meters range and 5.5-10 Hz scan rate[2].

In order to transform the coordinate systems of robot's links

[†] I built the robot with my team for the course DD2425 Robotics and Autonomous Systems, Autumn 2016. The hardware is supplied from the course. The course was with a team of four including me, however, things that are documented in this report (mechanical design, all the codes for localization, map server, teleoperation...) are completely implemented just by me.



(a) (b)

Figure 1 CAD model of the design 1a and the built robot 1b

a transform tree is created and a *transform broadcaster* published the coordinate transformations. This was needed to convert the laser scanner range data from sensor base to robot's base.

To test the robot in the maze a teleoperation node is created which reads from the keyboard and controls the motors for the given inputs using kinematics conversion and PID controllers. For this `teleop_twist_keyboard` package[3] is modified and `control_toolbox` pid controller[4] is used.

Particle Filter Localization

Particle Filter (PF) is a non-parametric Bayes filter[5] which estimates the posterior distribution with a number of hypotheses called *particles*. In the 2-dimensional localization case, each particle is a hypothesis of the robot's pose in 2D as (x, y, θ). There are *predict*, *update* and *resample* phases in PF.

The requirements from the localization system were high enough rate, being able to recover from errors (i.e. errors occurred from rubbing against walls) and to localize globally. As emphasized earlier, the latter comes naturally with PF and error recovery is a good metric to evaluate the PF performance.

The particle filter was implemented as a ROS node that *subscribes* to dead reckoning and laser scanner nodes, and *publishes* the current pose belief on a topic. Since it is needed a single pose value rather than a distribution to be used in other subsystems such as navigation or local map update, the published pose output was the mean (x, y, θ) values of all particles.

During the tasks of the robot, either global localization or just pose tracking was needed, depending on the prior pose knowledge of the robot. Hence two initialization choices were provided in the PF node. By setting a parameter in the ROS *parameter server*, it was possible to initialize the particles with either uniformly distributed within the given map so it would globally localize the robot, or normally distributed around a given initial pose.

The environment that robot was put was a maze with $30cm$ high white wooden walls forming corridors, rooms, shapes and obstacles. Different mazes were presented for robot to be tested. The map of maze environment was supplied in a format such that the end point coordinates of every line segment in the maze were given. One of the test mazes is shown in Figure 2, where an occupancy grid map is also created to be used in other applications such as path planning.

Predict

In the predict phase, a simple process model based on the information from encoders provided from dead reckoning node is used;

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + \Delta x \\ y_t + \Delta y \\ \theta_t + \Delta\theta \end{bmatrix} + \begin{bmatrix} N(0, \sigma_{linear}) \\ N(0, \sigma_{linear}) \\ N(0, \sigma_{angular}) \end{bmatrix} \quad (1)$$

The process noises on (x, y, θ) were zero mean Gaussian noises and they play a vital role in recovering from errors. The measurement uncertainty set higher on orientation because the errors on robot's pose became larger when it was rotating rather than moving straight.

Update

Update phase was the tricky part in implementing PF. It was very computationally expensive and in order to meet the rate requirements it was necessary to use some practical tricks. A measurement model was needed in order to get the probability of the current measurement given the state. Measurement model would then be used to calculate the weights for the particles. The more likely a particle is, the higher its weight should be.

For the measurement probability calculation, a beam model was used for the laser scanner. For each ray from the laser, a ray casting algorithm was applied to get the hit on the closest wall of the map, i.e. to *simulate* the laser scanner at the particle's pose. The measurement probability distribution was assumed

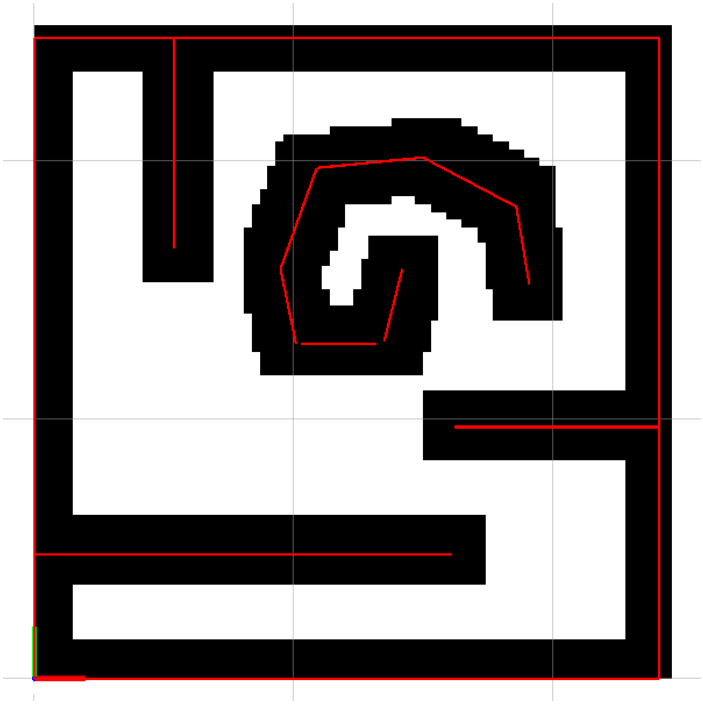


Figure 2 Map of one of the test mazes showed in ROS visualization interface *rviz*. Red lines are the walls in continuous space and black entries are the discrete dilated occupancy grid that corresponds to it. Map dimensions: $\sim 2.5 \times 2.5$ m

to be Gaussian, and using the differences in range between the measurements and simulated rays in each angle, it was calculated. The weight of a single particle was calculated as the mean instead of product of all the measurement probabilities of all the rays. This was to avoid having too small weights because of only one bad measurement or outliers. Then all the weights were normalized. Data association was not a problem since the angles of each range measurement were known.

The most time taking part was to simulate the laser rays for every particle which was done by analytically solving the line intersections for the rays at every angle to every wall in the map and getting the closest range. It was not possible to meet the rate requirements if ray casting was directly doing the analytical calculations. When considering the algorithm, it can be thought that there would be many line segments that do not intersect, but this could be only checked after analytically solving them and see if the solution lies within the segments' lengths. To check whether line segments would intersect before doing analytical solution and hence to make the calculations only on the line segments that intersect, several tests were made;

- **Checking Axis-Aligned Bounding Boxes (AABB):** It was very fast and straightforward to get and check AABBs of two line segments[6]. If two line segments are intersecting, their AABBs should collide. It is a rough, first check.
- **Moving segments and checking their endpoints‡:** If two line segments intersect, their endpoints should lie on the other sides of each other or at least one endpoint should be on top of the other. To check this, two line segments should be moved in Cartesian space such that endpoint of one segment is on the origin. When using cross product with end points, the sign of product would change according to the side of the endpoint[7].

Hence, after the tests, only the intersecting line segments were solved analytically and found the distance as range. Then the minimum range was picked to be the simulated scan on that angle.

Another way to reduce time in update phase was not to use 360 rays in every weight calculation. Instead scans were simulated for every 4 th or 8 th angle and corresponding measurements were used to calculate the weight.

Resample

In resampling phase, a new set of particles is created from the current set based on the weights of the particles. The probability of any particle to be resampled is proportional to its weight. This is like a survival of the fittest approach. The more probable the particle's state is based on the measurement, the higher its weight and thus the more likely it is to be resampled.

There are different approaches on how to do the resampling such as simple random resampling (vanilla/multinomial), stratified resampling and systematic resampling. Systematic resampling was chosen to be implemented since it has lower variance[8] and it is more precise than simple random and stratified random resampling[9].

Experiment

The PF localization system was tested in the maze environment for both global localization and pose tracking. The dead reckoning system was also kept tracked to be compared. In the experiment, 2000 particles and every 8th range measurement were used. With this setup, the mean estimation rate was a little more than 3 Hz. Full experiment video can be watched in https://youtu.be/ouPrylVx05I.

The robot was put in the maze with a random position and orientation and all the necessary data such as the particles, belief (mean of the particles), dead reckoning axis were visualized in ROS *rviz* and also recorded as *rosbag* files.

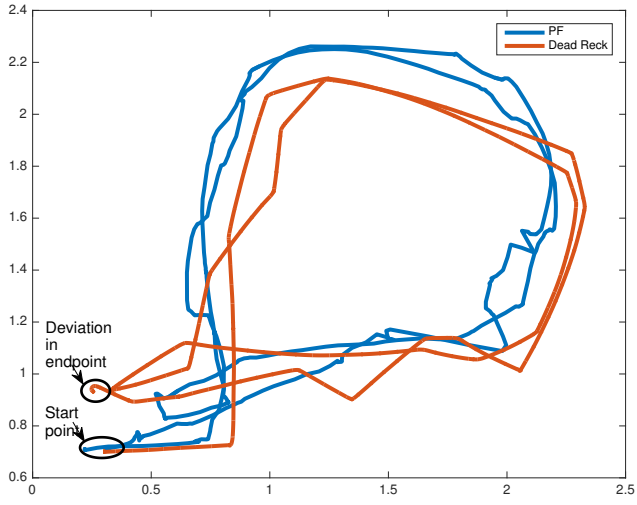


Figure 3 The path that robot followed in the experiment in correspondence with Figure 5, according to particle filter (blue) and dead reckoning system (red), axis units are in meters.

Initially, the particles were uniformly distributed in the maze as shown in Figure 4a and using the PF the robot successfully

‡ The algorithm by Martin Thoma[7] was modified and used in the source code.

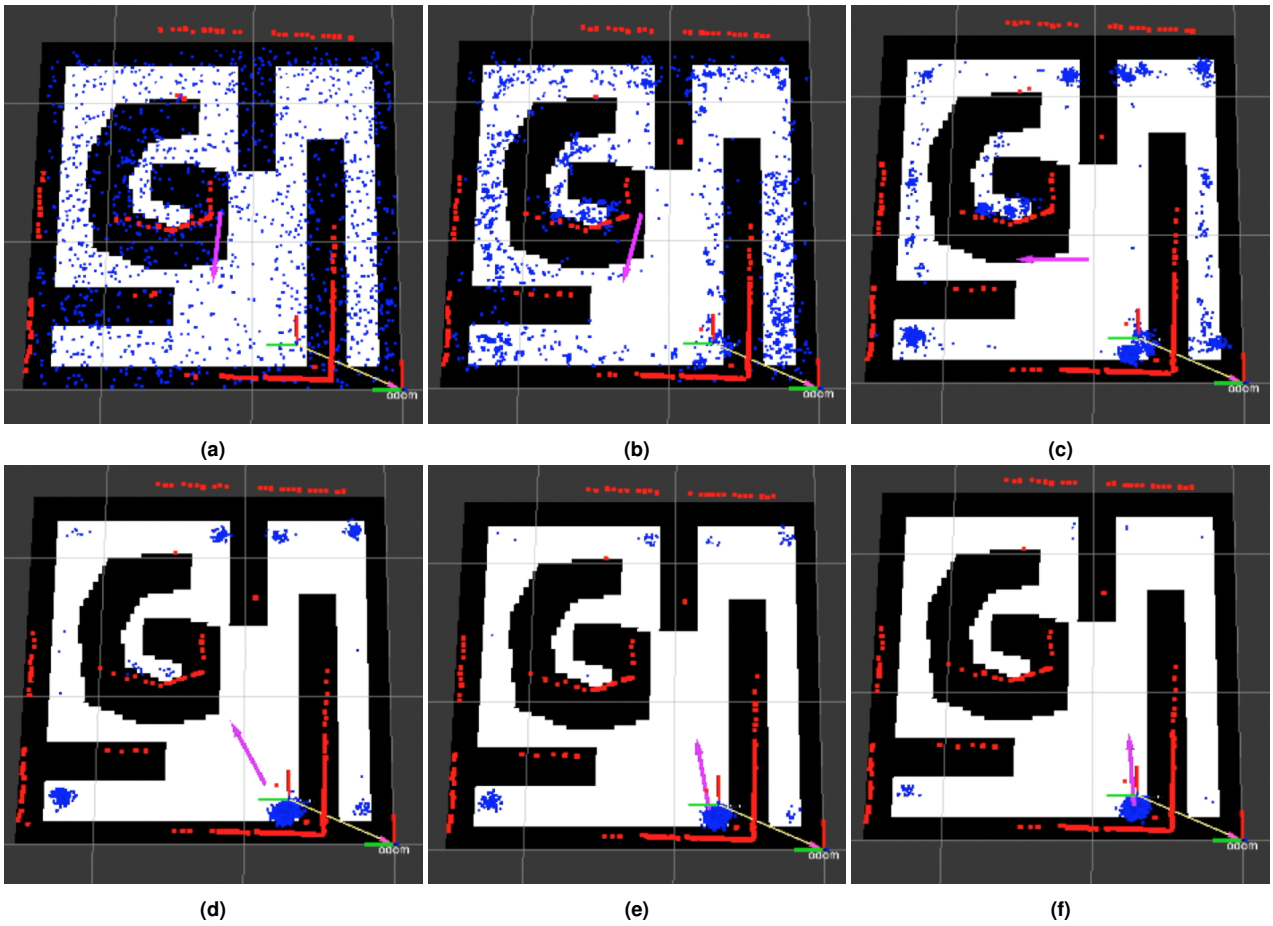


Figure 4 Robot globally localizing with uniform initial distribution of the particles. Blue dots →particles, purple arrow →belief (mean of the particles), red dots →laser sensor readings, red/green axes →robot's pose according to dead reckoning system

localized itself in the maze after some iterations (Figure 4). The convergence for the actual pose from the uniform distribution took approximately 10 seconds. In Figure 4, it can be seen that during the initial iterations the particles tend to populate the corners of the maze more. Since the robot is actually placed around a corner, the measurement model returns higher probability hence the weight for the particles around a corner. Then the false ones disappear. This test shows that the update and resample implementations are successfully working.

Once the robot localized itself, the robot is teleoperated in the maze to test the pose tracking. Tracking was successful as shown in Figure 5. The number of particles kept high for global localization, yet when the robot's initial pose is known and given, it was possible to successfully track the movement with even 100 particles.

Even though the dead reckoning system was working better than expected, the errors due to slippage were still significant as shown in Figure 3 where the path that robot moved during the experiment was plotted for both particle filter and dead reckoning estimations on top of each other. The skew in dead reckoning system can be clearly seen as well as deviation in the endpoint. During the experiment, the robot is moved around the maze twice in a loop and then went back to the starting point, which is in the bottom left point in Figure 3, close to the origin. The particle filter estimation was successfully returned to the starting point, however the dead reckoning system was deviated.

Conclusion

Prior to the particle filter implementation, I made a mechanical design for the robot and it was built with the help of my team in course DD2425. The essential applications such as motor control, teleoperation, robot setup on ROS and dead reckoning are done by me. Then I implemented the particle filter.

The localization could be initialized either for global localization or just tracking by only setting a parameter on the parameter server. An odometry based process model is used with additive zero mean Gaussian noise. The update part was tricky, a laser sensor scan was simulated by the code using ray casting algorithm and it was made faster using tricks to check line segment intersections prior to any calculation. Systematic resampling was chosen for the resampling method.

After the implementation, the localization system is tested on the robot with a wooden maze. The particle filter output rate was around 3 Hz with 2000 particles and every 8^{th} range measurement. It took approximately 10 seconds for robot to localize itself on the maze from a uniform distribution. Both global localization and tracking worked successfully.

Overall, the theory learned in the course is applied on a real hardware with real world implementation problems and restrictions. Particle filter is successfully implemented from scratch using C++ with no given skeleton, data set or test code.

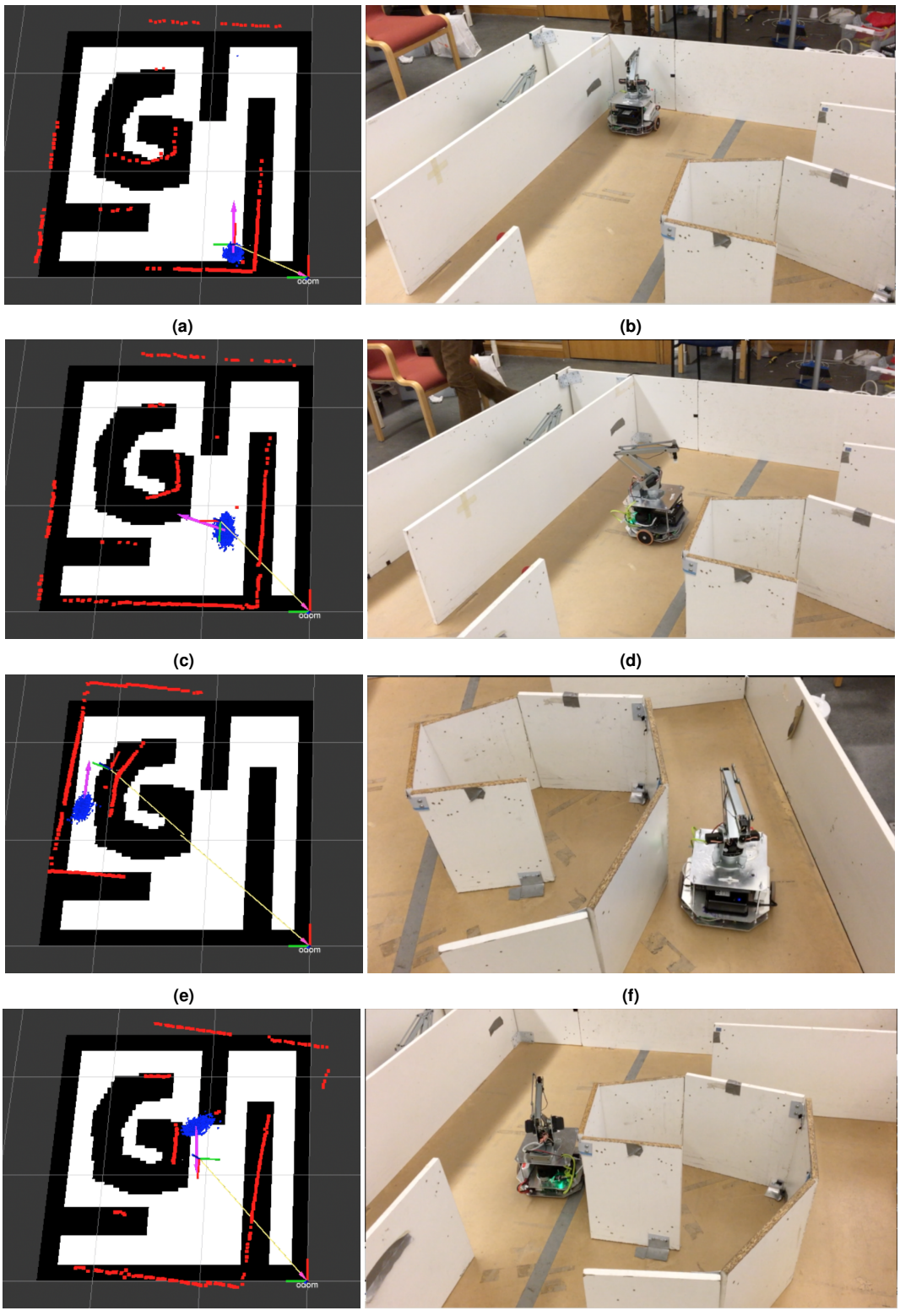


Figure 5 Pose tracking of the robot while it moves in the maze. Blue dots →particles, purple arrow →belief (mean of the particles), red dots →laser sensor readings, red/green axes →robot's pose according to dead reckoning system

Bibliography

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, International Conference on Robotics and Automation, 2009.

[2] Shanghai Slamtec.Co.,Ltd, *RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet Model: A1M8*, 2016.

[3] Graylin Trevor Jay, *ROS Teleop Twist Keyboard*, http://wiki.ros.org/teleop_twist_keyboard, Accessed: 07-12-2016.

[4] Melonee Wise, Sachin Chitta, John Hsu, *ROS Control Toolbox*, http://wiki.ros.org/control_toolbox, Accessed: 07-12-2016.

[5] S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics*, MIT Press, 2005.

[6] C. Ericson, *Real-Time Collision Detection*, Elsevier, 2005.

[7] M. Thoma, *How to check if two line segments intersect*, https://martin-thoma.com/how-to-check-if-two-line-segments-intersect/, Accessed: 01-11-2016.

[8] J. D. Hol, *Resampling in particle filters*, http://www.diva-portal.org/smash/get/diva2:19698/FULLTEXT01.pdf, 2004.

[9] W. G. Cochran, *Sampling Techniques*, Jonh Wiley and Sons, Inc., 1977.