Monte Carlo Localization for teach-and-repeat feature-based navigation

Matías Nitsche[1], Taihú Pire[1], Tomáš Krajník[2], Miroslav Kulich[3], and Marta Mejail[1]

[1] Laboratory of Robotics and Embedded Systems, Computer Science Department, Faculty of Exact and Natural Sciences, University of Buenos Aires
[2] Lincoln Centre for Autonomous Systems, School of Computer Science, University of Lincoln, UK
[3] Intelligent and Mobile Robotics Group, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague
`mnitsche@dc.uba.ar, tpire@dc.uba.ar, tkrajnik@lincoln.ac.uk, kulich@labe.felk.cvut.cz, marta@dc.uba.ar`

Abstract. This work presents a combination of a teach-and-replay visual navigation and Monte Carlo localization methods. It improves a reliable teach-and-replay navigation method by replacing its dependency on precise dead-reckoning by introducing Monte Carlo localization to determine robot position along the learned path. In consequence, the navigation method becomes robust to dead-reckoning errors, can be started from at any point in the map and can deal with the 'kidnapped robot' problem. Furthermore, the robot is localized with MCL only along the taught path, i.e. in one dimension, which does not require a high number of particles and significantly reduces the computational cost. Thus, the combination of MCL and teach-and-replay navigation mitigates the disadvantages of both methods. The method was tested using a P3-AT ground robot and a Parrot AR.Drone aerial robot over a long indoor corridor. Experiments show the validity of the approach and establish a solid base for continuing this work.

1 Introduction

The problem of autonomous navigation has been widely addressed by the mobile robotics research community. The problems of localization and mapping, which are closely related to the navigation task, are often addressed using a variety of approaches. The most popular is Simultaneous Localization and Mapping[1] (SLAM), with successful examples such as MonoSLAM[2] and more recently PTAM[3], which uses vision as the primary sensor. However, these approaches generally model both the environment and the pose of the robot with a full metric detail, using techniques such as Structure From Motion (SfM) or stereo-based reconstruction[4]. While successful, these methods are computationally demanding and have shortcomings which arise from the complexity of the mathematical models applied.

In terms of solving the navigation problem, it may not always be necessary to obtain a detailed metric description of the environment nor to obtain a full 6DoF pose. For example, teach-and-replay (T&R) techniques[5,6,7,8,9,10], where the robot is supposed to autonomously navigate a path that it learned during a tele-operated run, do not require to explicitly localize the robot.

While standard visual navigation approaches depend on a large number of correspondences to be established to estimate the robot position, feature-based T&R methods are robust to landmark deficiency, which makes them especially suitable for scenarios requiring long-term operation [11]. Moreover, the T&R methods are generally much simpler and computationally less-demanding. On the other hand, they are usually not able to solve the kidnapped-robot problem nor to localize a robot globally, requiring knowledge about the initial robot position. Also, most of them are susceptible to errors in dead-reckoning caused by imprecise sensors or wheel slippage.

To address these shortcomings, the use of particle-filters such as Monte Carlo Localization (MCL)[12] becomes attractive. MCL is able to efficiently solve the localization problem and addresses the global-localization and kidnapped-robot problems, and has been applied with success in vision-based applications[13,14] and robot navigation[6,10,15].

In this work, an existing visual T&R method[7,8] is improved by replacing localization by means of dead-reckoning with Monte Carlo Localization. The original method relied on the precision of it's dead reckoning system to determine the distance it has travelled from the path start. The travelled distance estimation determines the set of features used to calculate the robot lateral displacement from the path and thus influences the precision of navigation significantly. While the authors show [7] that estimating the distance with dead-reckoning is sufficient for robots with precise odometry and paths that do not contain long straight segments, these conditions might not be always met. By replacing this dead-reckoning with MCL, the localization error can be diminished, while not only tackling the kidnapped-robot problem but also allowing the navigation to start at an arbitrary point in the taught path.

2 Related work

Teach-and-replay methods have been studied in several works. While some approaches are based on metric reconstruction techniques, such as stereo-based triangulation[9,16], several works employ so-called appearance-based or qualitative navigation[5,17,18].

This type of navigation uses simple control laws[5] to steer the robot according to landmarks remembered during a training phase. While some works approach the localization problem with image-matching techniques[19,20,21], others approaches use only local image features[18,17]. While the feature-based techniques are generally robust and reliable, image feature extraction and matching can be computationally demanding. Thus, strategies for efficient retrieval of landmarks are explored, such as with the LandmarkTree-Map[18].

However, feature- or dead-reckoning- based localization is error-prone and several visual T&R navigation methods propose to improve it by using advanced techniques such as Monte Carlo Localization (MCL). One of these works corresponds to [15] where MCL is applied together with an image-retrieval system to find plausible localization hypotheses. Another work[6] applies MCL together with FFT and cross-correlation methods in order to localize the robot using an omni-directional camera and odometry. This system was able to robustly repeat a 18 km long trajectory. Finally, a recent work[10] also employs MCL successfully with appearance-based navigation.

Based on similar ideas, the method proposed in this work applies MCL to a visual T&R navigation method[7] that proved its reliability in difficult scenarios including life-long navigation in changing environments[11]. Despite of simplicity of the motion and sensor models used in this work's MCL, the MCL proven to be sufficient for correct localization of the robot. In particular, and as in [10,6], the localization only deals with finding the distance along each segment of a topological map and does not require determining orientation. However, in contrast to the aforementioned approaches, global-localization and the kidnapped-robot problem are considered in our work.

3 Teach-and-replay method

In this section, the original teach-and-replay algorithm, which is improved in the present work, is presented. During the *teaching* or mapping step, a feature-based map is created by means of tele-operation. The *replay* or navigation phase uses this map to repeat the learned path as closely as possible. For further details about the method please refer to works[7,8].

3.1 Mapping

The mapping phase is carried out by manually steering the robot in a turn-move manner. The resulting map thus consists of a series of linear segments, each of a certain length and orientation relative to the previous one. To create a map of a segment, salient image-features (STAR/BRIEF) from the robot's onboard camera images are extracted and tracked as the robot moves. Once a tracked feature is no longer visible, it is stored as a landmark in the current segment's map. Each landmark description in the map consists of its positions in an image and the robot's distance relative to the segment start, both for when the landmark is first and last seen. Finally, the segment map contains the segment's length and orientation estimated by dead-reckoning.

In the listing 1, the segment mapping algorithm is presented in pseudo-code. Each landmark has an associated descriptor l_{desc}, pixel position l_{pos_0}, l_{pos_1} and robot relative distance l_{d_0}, l_{d_1}, for when the landmark was first and last seen, respectively.

Algorithm 1: Mapping Phase

```
Input: $F$: current image features, $d$: robot distance relative to segment start, $T$:
       landmarks currently tracked
Output: $L$: landmarks learned current segment
foreach $l \in T$ do
    $f \leftarrow$ find_match($l$,$F$)
    if no match then
        $T \leftarrow T - \{ l \}$                                          /* stop tracking */
        $L \leftarrow L \cup \{ l \}$                                       /* add to segment */
    else
        $F \leftarrow F - \{ f \}$
        $t \leftarrow (l_{desc}, l_{pos_0}, f_{pos}, l_{d_0}, d)$ /* update image coordinates & robot position */
foreach $f \in F$ do
    $T \leftarrow T \cup \{ (f_{desc}, f_{pos_0}, f_{pos_0}, d, d) \}$      /* start tracking new landmark */
```

3.2 Navigation

During the navigation phase, the robot attempts to repeat the originally learned path. To do so, starting from the initially mapped position, the robot moves forward at constant speed while estimating its traveled distance using dead-reckoning. When this distance is equal to the current segment's length, the robot stops and turns in the direction of the following segment and re-initiates the forward movement. To correct for lateral deviation during forward motion along a segment, the robot is steered so that that its current view matches the view perceived during the training phase. To do so, it continuously compares the features extracted from the current frame to the landmarks expected to be visible at the actual distance from the segment's start.

The listing 2 presents the algorithm used to traverse or 'replay' one segment. Initially, the list T of expected landmarks at a distance d from segment start is created. Each landmark $l \in T$ is matched to features F of the current image. When a match is found, the difference between the feature's pixel position f_{pos} is compared to an estimate of the pixel-position of l at distance d (obtained by linear interpolation). Each of these differences is added to a histogram H. The most-voted bin, corresponding to the mode of H, is considered to be proportional to the robot's lateral deviation and is therefore used to set the angular velocity ω of the robot. Thus, the robot is steered in a way that causes the mode of H to be close to 0. Note that while for ground robots the horizontal pixel-position difference is used, it is possible to apply it also with vertical differences, which allows to deploy the algorithm for aerial robots[8] as well.

While only lateral deviations are corrected using visual information, it can be mathematically proven[7] that the error accumulated (due to odometric errors) in the direction of the previously traversed segment is gradually reduced if the currently traversed segment is not collinear with the previous one. In practice, this implies that if the map contains segments of different orientations, the robot position error during the autonomous navigation is bound.

Algorithm 2: Navigation Phase

```
Input: $L$: landmarks for present segment, $s_{length}$: segment length, $d$: current
       robot distance from segment start
Output: $\omega$: angular speed of robot
while $d < s_{length}$ do
    $H \leftarrow \emptyset$                                   /* pixel-position differences */
    $T \leftarrow \emptyset$                                   /* tracked landmarks */
    foreach $l \in L$ do
        if $l_{d_0} < d < l_{d_1}$ then
            $T \leftarrow \cup \{ l \}$                        /* get expected landmarks according to $d$ */
    while $T$ not empty do
        $f \leftarrow$ find_match($l$,$F$)
        if matched then
            /* compare feature position to estimated current landmark position by
               interpolation                                                  */
            $h \leftarrow f_{pos} - \left( (l_{pos_1} - l_{pos_0}) \frac{d - l_{d_0}}{l_{d_1} - l_{d_0}} + l_{pos_0} \right)$
            $H \leftarrow \{ h \}$
        $T \leftarrow T - \{ l \}$
    $\omega \leftarrow \alpha$ mode($H$)
```

However, this restricts the method to maps with short segments whenever significant dead-reckoning errors are present (eg. wheel slippage). Otherwise, it is necessary to turn frequently to reduce the position error [22]. Moreover, the method assumes that the autonomous navigation is initiated from a known location. This work shows that the aforementioned issues are tackled by applying MCL.

4 Monte Carlo visual localization

In the present work the position estimate d of the robot relative to the segment start is not obtained purely using dead-reckoning. Rather than that it is obtained by applying feature-based MCL. With MCL, if sufficient landmark matches can be established, the error of the position estimation will remain bounded even if a robot with poor-precision odometry will have to traverse a long segment. Moreover, the MCL is able to solve the global localization problem which allows to start the autonomous navigation from arbitrary locations along the taught path.

The robot localization problem consists in finding the robot state $\mathbf{x}_k$ at time k given the last measurement z^k (Markov assumption) and a prior state $\mathbf{x}_{k-1}$. MCL, as any bayesian-filter based approach, estimates the probability density function (PDF) $p(\mathbf{x}_k|z^k)$ recursively, i.e. the PDF calculation uses information about the PDF at time $k-1$. Each iteration consists of a *prediction* phase, where a motion model is applied (based on a control input u_{k-1}) to obtain a predicted PDF $p(\mathbf{x}_k|z^{k-1})$, and an *update* phase where a measurement z^k is applied to obtain an estimate of $p(\mathbf{x}_k|z^k)$. Under the MCL approach, these PDFs are represented by means of samples or *particles* that are drawn from the estimated probability distribution that represents the robot's position hypothesis.

In the present work, since the goal of the navigation method is not to obtain an absolute 3D pose of the robot, the state vector is closely related to the type of map used. In other words, since the environment is described by using a series of segments which contain a list of tracked landmarks along the way, the robot state $\mathbf{x}$ is simply defined in terms of a given segment s_i and a robot distance from the segment's start d:

$$\mathbf{x} = [s_i \; d]$$

. The orientation and lateral displacement of the robot is not modeled in the state since these are corrected using the visual input. This simplification leads to a low dimensional search-space that does not require to use a large number of particles.

The MCL algorithm starts by generating m particles p_k uniformly distributed over the known map. As soon as the robot starts to autonomously navigate, the prediction and update phases are continuously applied to each particle.

4.1 Prediction step

The prediction step involves applying a motion-model to the current localization estimate represented by the current value of particles p_k, by sampling from $p(\mathbf{x}_k|\mathbf{x}_{k-1}, u_k)$, where u_k is the last motion command. The motion model in this work is defined as:

$$f([s_i \; d]) = \begin{cases} [s_i, d + \Delta\ d + \mu] & (d + \Delta d) < \text{length}(s) \\ [s_{i+1}, (d + \Delta d + \mu) - \text{length}(s)] & \text{else} \end{cases}$$

where Δd is the distance traveled since the last prediction step and is a random variable with Normal distribution, i.e. $\mu \sim \mathcal{N}(0, \sigma)$. The value of σ was chosen as 10 cm. This noise term is necessary to avoid premature convergence (i.e. a single high-probability particle chosen by MCL) and to maintain the diversity of localization hypotheses. Finally, this term can also account for the imprecision of the motion model which does not explicitly account for the robot heading.

The strategy to move particles to the following segment, which is also used in other works[10], may not necessarily be realistic since it does not consider the fact that the robot requires rotating to face the following segment. However, not propagating the particles to the following segment means that the entire particle set needs to be reinitialized every time new segment traversal is started, which is impractical.

4.2 Update step

In order to estimate $p(\mathbf{x}_k|z^k)$, MCL applies a sensor model $p(z_k|\mathbf{x}_k)$ to measure the relative importance (weight) of a given particle p_i. Particles can then be resampled considering these weights, thus obtaining a new estimate of $\mathbf{x}_k$ given the last measurement z^k. In this work, z_k corresponds to the features F extracted at time k. The weight w_i of a particle p_i is computed in this work as:

$$w_i = \frac{\#\text{matches}(F, T)}{|T|}$$

where T is the set of landmarks expected to be visible according to p_i. In other words, the weight of a particle is defined as the ratio of visible and expected landmarks. The weights w_i are then normalized in order to represent a valid probability $p(z_k|p_i)$. It should be noted that it is possible for a particle to have $w_i = 0$ (ie. no features matched). Since every particle requires to have a non-zero probability of being chosen (i.e. no particles should be lost), particles with $w_i = 0$ are replaced by particles generated randomly over the entire map.

While this sensor model is simple, the experiments proved that it was sufficient to correctly match a particle to a given location.

4.3 Re-sampling

To complete one iteration of MCL, a new set of particles is generated from the current one by choosing each particle p_i according to its weight w_i. A naive approach for this would be to simply choose each particle independently. However, a better option is to use the low-variance re-sampling algorithm [1]. In this case, a single random number is used as a starting point to obtain M particles honoring their relative weights. Furthermore, this algorithm is of linear complexity.

The low-variance sampler is used not only to generate particles during the update step, but also to generate uniformly distributed particles in the complete map (during initialization of all particles and re-initialization of zero-weight particles). In this case, the set of weights used consists of the relative length of each segment in the map.

4.4 Single hypothesis generation

While MCL produces a set of localization hypotheses, the navigation needs to decide which hypothesis is the correct one. Furthermore, to asses the quality of this type of localization compared to other methods (such as a simple dead-reckoning approach) it is necessary to obtain a single hypothesis from the complete set.

To this end, the proposed method to obtain a single hypothesis is to compute the mean of all particle positions over the complete map. This is the usual choice for uni-modal distributions but is not suitable in general for an approach such as MCL which may present multiple modalities. However, when MCL converges to a single hypothesis, the mean position of all particles is a good estimate for the true position of the robot. In these cases, all particles are found to be clustered around a specific region and the population standard deviation is similar to the σ value used as the motion model noise. Therefore, it is possible to check if the standard deviation of all particles is less than $k\sigma$ and if so, the mean position will be a good estimate of the true robot position. In other cases, if the standard deviation is higher, the mean may not be a good position estimate. This can happen when the robot is "lost" and when the underlying distribution is multi-modal (i.e. when the environment is self-similar and the current sensor readings do not uniquely determine the robot position).

Finally, since an iteration of MCL can be computationally expensive and single-hypothesis estimates can only be trusted whenever the std. dev. is small enough, dead-reckoning is used to update the last good single-hypothesis whenever MCL is busy computing or it has failed to find a good position estimate.

5 Experiments

The proposed system was implemented in C/C++ within the ROS (Robot Operating System) framework as a set of separate ROS modules. In order to achieve high performance of the system, each module was implemented as a ROS *nodelet* (a thread).

Tests were performed over a long indoor corridor using a P3-AT ground and a Parrot AR.Drone aerial robot. The P3-AT's training run was $\sim 100\,\text{m}$ long while the AR-Drone map consisted of a $\sim 70\,\text{m}$ long flight. The P3-AT used wheel encoders for dead-reckoning and a PointGrey 0.3MP FireflyMV color camera configured to provide 640×480 pixel images at $30\,\text{Hz}$. The AR.Drone's dead reckoning was based on its bottom camera and inertial measurement unit and images were captured by its forward camera that provides 320×240 pixel images at $17\,\text{Hz}$. A Laptop with a quad-core Corei5 processor running at $2.3\,\text{GHz}$ and $4\,\text{GB}$ of RAM was used to control both of the robots.

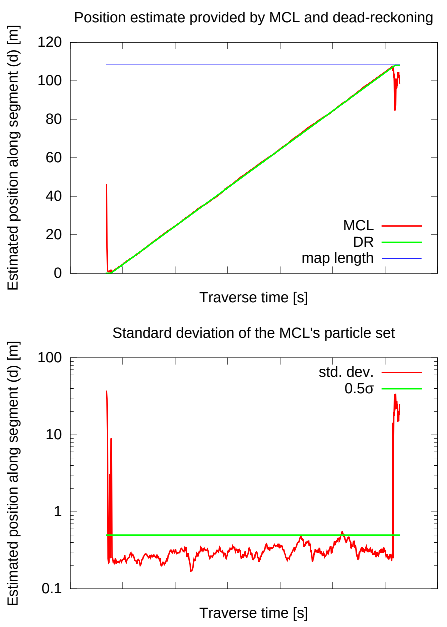
Both robots were manually guided along the corridor three times while recording images and dead-reckoning data. For each robot, one of the three datasets was used for an off-line training run and the remaining ones were used for the replay phase evaluation (we refer to the remaining two datasets as 'replay 1' and 'replay 2'). Since this work focuses on the use of MCL for improving the original teach-and-replay method, only the localization itself is analyzed. Nevertheless, preliminary on-line experiments with autonomous navigation were performed using MCL which yielded promising results.[4]

In figure 1 the position-estimates of the robot obtained by dead-reckoning and MCL are compared. The standard deviation of the particle population in relation to the threshold of 5σ is also presented, see 1.

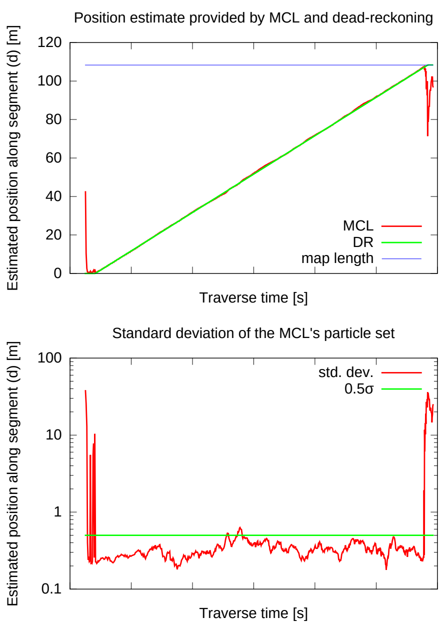
When analyzing figures 1(a),1(b), it becomes evident that the odometry of the P3AT is very precise and MCL does not provide a significant improvement. The standard deviations (see figures 1(a),1(b)) for these two cases show an initial delay caused by initialization of the MCL. Once the MCL particles converge, the position estimate becomes consistent and remains stable during the whole route.

On the other hand, in the AR.Drone localization results (see figures 1(c),1(d)) there's a noticeable difference between the MCL and dead-reckoning estimates due to the imprecision of the AR.Drone's dead-reckoning. During the 'replay 2',

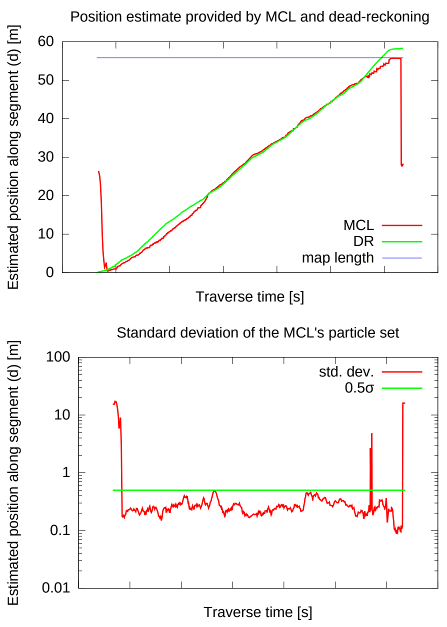
[4] Note to reviewers: experiments are in progress, more exhaustive results will be ready for the camera-ready version



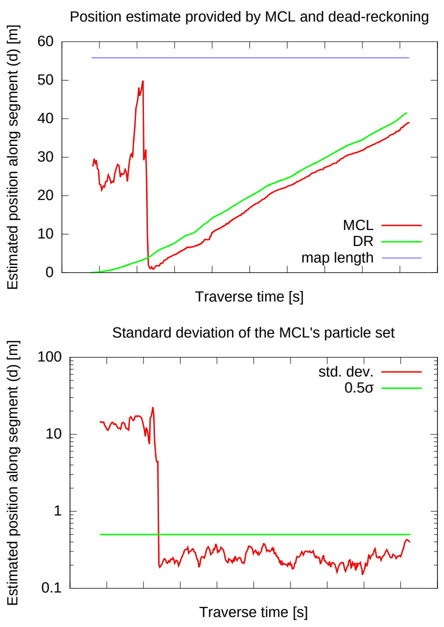
(a) P3-AT, replay 1



(b) P3-AT, replay 2



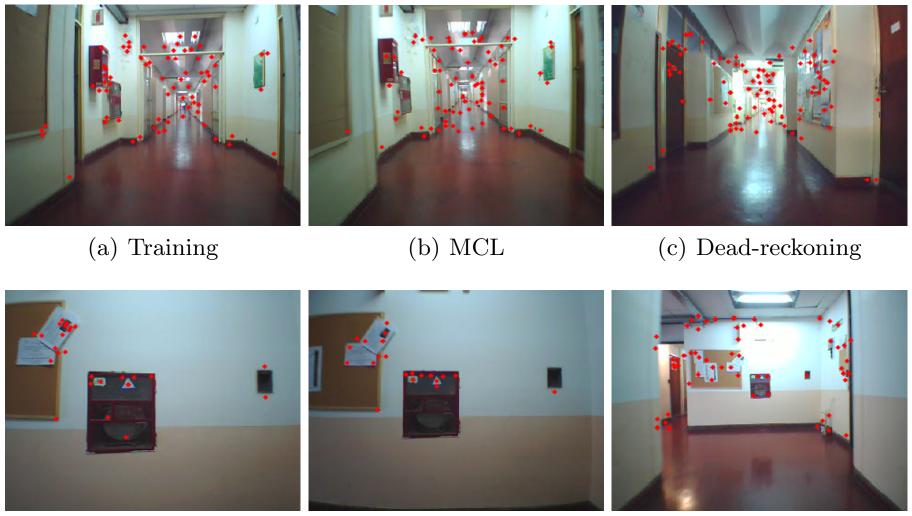
(c) AR.Drone, replay 1



(d) AR.Drone, replay 2

Fig. 1. Comparison of position estimates obtained from dead-reckoning (DR) and MCL, with corresponding standard deviation of the MCL's particle set.

the AR.Drone was placed a few meters before the initial mapping point, which is reflected by MCL, but not by the dead-reckoning, see Figure 1(d).



(a) Training (b) MCL (c) Dead-reckoning

(d) Training (e) MCL (f) Dead-reckoning

Fig. 2. Parrot AR.Drone view during replay and training, for matching distance estimates obtained with MCL and dead-reckoning: **(a)-(c)**: replay 2, $d \approx 0$; **(d)-(f)**: replay 1, $d \approx \text{length}(s)$

When analyzing the AR.Drone's 'replay 1' (see Figure 1(c)) it can be seen that after the initial convergence of MCL, the position estimates differ only slightly. However, there is a particular difference at the segment end where the drone actually overshot the ending point. In this case, MCL correctly estimates that the robot has reached the end, while with dead-reckoning the estimate goes beyond the segment length. Eventually, the MCL estimate diverges but this happens since the robot landed very close to a wall and suddenly no more recognizable feature were visible.

A similar situation occurred on 'replay 2' (1(d)) where the initial displacement of the AR.Drone is corrected by MCL and its difference to the odometry remains prominent. In this case, the MCL did not offer a suitable position hypothesis until the drone reached the segment start (notice the high standard deviation of the particles on 1(d)). One can see that while the MCL position estimate converges to 0 as soon as the robot passes the segment start, the position estimate provided by the dead-reckoning retains the offset throughout the entire segment.

Since no ground-truth is available to verify the correctness of the position estimates, we present the images taken when the MCL or dead-reckoning reported the drone to be at the segment start or end, see figures 2(a)-2(c) and 2(d)-2(f).

It can be seen that in the case of MCL position estimation, the images captured by the robot are more similar to the images seen during the training phase. In the case of dead-reckoning, the views differ from the ones captured during the training phase which indicates a significant localization error.

6 Conclusions

This article has presented a combination of a teach-and-replay method and Monte Carlo Localization. The Monte Carlo Localization is used to replace the dead-reckoning-based position estimation that prohibited to use the teach-and-replay method by robots without a precise odometric system or in environments with long straight paths. Moreover, the ability of the MCL to deal with the kidnapped robot problem allows to initiate the 'replay' (i.e. autonomous navigation) from any location along the taught path. The experiments have shown that MCL-based position estimate remains consistent even in situations when the robot is displaced from the path start or its odometric system is imprecise.

To further elaborate on MCL's precision, we plan to obtain ground-truth position data by using an external localization system introduced in [23], In the future, we plan to improve the method by more robust calculation of the MCL's position hypothesis, more elaborate sensor model and improved feature extraction (eg. fixed number of features per frame). We also consider approaches such as Mixture Monte Carlo [24], that may provide further advantages.

Finally, the use of MCL opens the possibility of performing loop-detection, which will allow to create real maps rather than just remember visual routes. Using loop-detection, the robot would automatically recognize situations when the currently learned segment intersects the already taught path. Thus, the user will not be required to explicitly indicate the structure of the topological map that is learned during the tele-operated run.

Acknowledgements

The work is supported by the EU project 600623 'STRANDS' and the Czech-Argentine bilateral cooperation project ARC/13/13 (MINCyT-MEYS).

References

1. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT Press (2005)
2. Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: MonoSLAM: real-time single camera SLAM. IEEE transactions on pattern analysis and machine intelligence **29** (2007) 1052–1067
3. Klein, G., Murray, D.: Parallel Tracking and Mapping on a camera phone. 2009 8th IEEE International Symposium on Mixed and Augmented Reality (2009)
4. Hartley, R., Zisserman, A.: Multiple view geometry in computer vision. (2003)
5. Chen, Z., Birchfield, S.: Qualitative vision-based mobile robot navigation. In: Robotics and Automation, ICRA. Proceedings International Conference on, IEEE (2006) 2686–2692

6. Zhang, A.M., Kleeman, L.: Robust appearance based visual route following for navigation in large-scale outdoor environments. The International Journal of Robotics Research **28**(3) (2009) 331–356
7. Krajník, T., Faigl, J., Vonásek, V., Kosnar, K., Kulich, M., Preucil, L.: Simple yet stable bearing-only navigation. Journal of Field Robotics **27**(5) (2010) 511–533
8. Krajnik, T., Nitsche, M., Pedre, S., Preucil, L.: A simple visual navigation system for an UAV. In: Systems, Signals and Devices. (2012) 1–6
9. Furgale, P., Barfoot, T.: Visual teach and repeat for longrange rover autonomy. Journal of Field Robotics (2006) (2010) 1–27
10. Siagian, C., Chang, C.K., Itti, L.: Autonomous Mobile Robot Localization and Navigation Using a Hierarchical Map Representation Primarily Guided by Vision. Journal of Field Robotics **31**(3) (May 2014) 408–440
11. Krajník, T., Pedre, S., Přeučil, L.: Monocular Navigation System for Long-Term Autonomy. In: Proceedings of the International Conference on Advanced Robotics, Montevideo, IEEE (2013)
12. Dellaert, F., Fox, D., Burgard, W., Thrun, S.: Monte Carlo localization for mobile robots. IEEE International Conference on Robotics and Automation **2** (1999)
13. Niu, M., Mao, X., Liang, J., Niu, B.: Object tracking based on extended surf and particle filter. In: Intelligent Computing Theories and Technology. Volume 7996 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 649–657
14. Li, R.: An Object Tracking Algorithm Based on Global SURF Feature. Journal of Information and Computational Science **10**(7) (May 2013) 2159–2167
15. Wolf, J., Burgard, W., Burkhardt, H.: Robust vision-based localization by combining an image-retrieval system with monte carlo localization. Transactions on Robotics **21**(2) (April 2005) 208–216
16. Ostafew, C.J., Schoellig, A.P., Barfoot, T.D.: Visual teach and repeat, repeat, repeat: Iterative Learning Control to improve mobile robot path tracking in challenging outdoor environments. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (November 2013) 176–181
17. Ok, K., Ta, D., Dellaert, F.: Vistas and wall-floor intersection features-enabling autonomous flight in man-made environments. Workshop on Visual Control of Mobile Robots (ViCoMoR): IEEE/RSJ International Conference on Intelligent Robots and Systems (2012)
18. Augustine, M., Ortmeier, F., Mair, E., Burschka, D., Stelzer, A., Suppa, M.: Landmark-Tree map: A biologically inspired topological map for long-distance robot navigation. Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on (2012) 128–135
19. Ni, K., Kannan, A., Criminisi, A., Winn, J.: Epitomic location recognition. IEEE transactions on pattern analysis and machine intelligence **31**(12) (2009) 2158–67
20. Cadena, C., McDonald, J., Leonard, J., Neira, J.: Place recognition using near and far visual information. Proceedings of the 18th IFAC World Congress (2011)
21. Matsumoto, Y., Inaba, M., Inoue, H.: Visual navigation using view-sequenced route representation. In: Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on. Volume 1. (Apr 1996) 83–88 vol.1
22. Faigl, J., Krajník, T., Vonásek, V., Preucil, L.: Surveillance planning with localization uncertainty for uavs. 3rd Israeli Conference on Robotics, Ariel (2010)
23. Krajník, T., Nitsche, M., Faigl, J., Vank, P., Saska, M., Peuil, L., Duckett, T., Mejail, M.: A practical multirobot localization system. Journal of Intelligent and Robotic Systems (2014) 1–24
24. Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust Monte Carlo localization for mobile robots. Artificial Intelligence **128** (2001) 99–141