Orbiting a Moving Target with Multi-Robot Collaborative Visual SLAM

Jacob M. Perron*, Rui Huang*, Jack Thomas, Lingkang Zhang, Ping Tan, and Richard T. Vaughan.
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
{jperron, rha55, jackt, lza90, pingtan, vaughan}@sfu.ca

***Abstract*—Towards autonomous 3D modelling of moving targets, we present a system where multiple ground-based robots cooperate to localize, follow and scan from all sides a moving target. Each robot has a single camera as its only sensor, and they perform collaborative visual SLAM (CoSLAM). We present a simple robot controller that maintains the visual constraints of CoSLAM while orbiting a moving target so as to observe it from all sides. Real-world experiments demonstrate that multiple ground robots can successfully track and scan a moving target.**

I. INTRODUCTION

We are working towards an autonomous robot system that can acquire live 3D reconstructions of people moving outdoors. This would be a widely useful capability with applications in entertainment, recreation, security and health.

This paper reports on our pilot study, where we investigate the feasibility of performing this task with a team of robots using cooperative monocular SLAM as the only sensor. We use slow-moving, low-cost non-holonomic ground robots with generic web-cam sensors and slow-moving but non-cooperative targets.

Existing vision-based collaborative SLAM systems [1] [9] provide a means for localizing robots but are not suitable for target following because they do not handle dynamic environments. Other visual approaches to dynamic object detection involve template matching or machine learning techniques. These methods usually require instrumentation or extensive training that limits their applicability.

By contrast, CoSLAM [14] uses multiple camera views to segment moving objects from the stationary background. This provides us a means to detect and track targets. CoSLAM requires certain constraints on the set of camera views, and our our goal application requires that we observe the target from all sides as often as possible. We describe a multi-robot system that scans non-cooperative moving targets by employing a simple motion controller that causes the robots to orbit the target while accommodating CoSLAM's visual constraints. We validate the approach with real-world experiments.

The contributions of this paper are: (i) the first demonstration of multi-robot visual-SLAM-only target following and scanning; (ii) a description of the improvements made to CoSLAM to improve robustness and permit real-time operation; (iii) description of a simple motion controller that tracks and orbits targets while respecting SLAM camera pose constraints.

*These authors contributed equally to this work.

Fig. 1: Two robots following and orbiting a human using CoSLAM as the only sensor.

II. RELATED WORK

Our system combines collaborative SLAM and multi-robot target following.

A. Localization using Multiple Cameras

Recent work uses multiple cameras to collaboratively localize robots. Chang et al. [5] implement a vision based localization and tracking system for Nao robots in a soccer field environment. The system is limited to 2D environments and shape detection is used to track the other robots and the ball. Dhiman et al. [7] use reciprocal observations of fiducial markers for the camera pose estimation. Their approach is not designed for dynamic scenes and the fiducial markers need to be present in the image views. Dias et al. [8] proposes an uncertainty based multi-robot cooperative triangulation method for target position estimation. However, they did not solve the camera poses and target detection problem. In [9], multiple UAVs perform monocular visual SLAM and estimate their motion individually. A centralized ground station receives preprocessing results and creates a global coordinate frame. Due to the lack of cooperation between cameras, the system is not able to handle dynamic environments.

CoSLAM is a recently-developed cooperative visual SLAM method, which operates by receiving video frames from independent moving cameras. By cooperatively tracking and creating a global 3D map, multiple independent cameras are

able to detect and track dynamic feature points in real time.

CoSLAM was designed and previously demonstrated for offline processing of pre-recorded, synchronized HD videos. Since it needs only off-the-shelf color cameras as sensor input, it is suitable for platforms with limited payload such as mobile robots. This paper is the first demonstration of an improved, more robust CoSLAM implementation that is generalized to work in real time with online video streams.

B. Target Following

Robot controllers for target tracking have been widely studied under different settings in robotics. Dang et al. [6] track a moving target with multiple UAVs in a formation anchored to a single leader. In [10], multiple robots localize and encircle a target in a distributed manner with range-finders. Aranda et at. [2] propose a general method for enclosing a target with any 3D geometric formation. Unlike the aforementioned approaches, our controller must consider the camera sensor constraints in order to locate the target. Yang et al.'s [13] work is similar to ours in that the robots move in order to achieve a desired sensor reading.

III. System Design

To solve the problem of generic target following our system consists of two components: CoSLAM and the robot controller. The robots each run a mono camera mounted orthogonal to its heading (90 degrees counterclockwise here) and an on-board controller. CoSLAM runs on a base station receiving video frames from each robot to produce pose estimations for all robots and the target. Communications are by commodity WiFi. As CoSLAM was developed with offline processing of pre-recorded high-quality videos that are synchronized manually, there are difficulties in applying it to platforms with limited sensing payload in real time applications. First, we discuss CoSLAM and modifications made to the previous system followed by details about the robot controller.

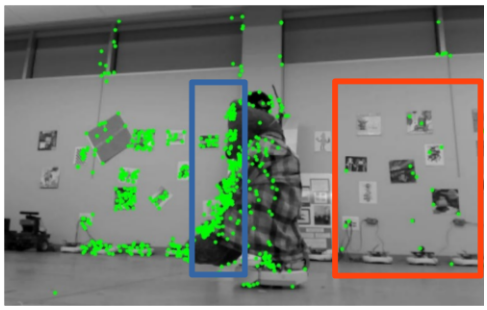
A. CoSLAM

Following the conventional sequential structure-from-motion (SFM) pipeline, the CoSLAM system uses multiple cameras as sensor inputs. Each camera is calibrated beforehand and can move independently. At the initialization stage, all the cameras are required to look at the same scene to construct a global 3D map. After initialization, the cameras start tracking their poses by registering the 2D image features to the corresponding map points. The 2D features for an individual camera are detected and tracked using a Kanade-Lucas-Tomasi (KLT) [12] tracker for the sake of efficiency. When the number of visible map points drops significantly, new map points can be generated using the feature matches from a single camera or multiple cameras.

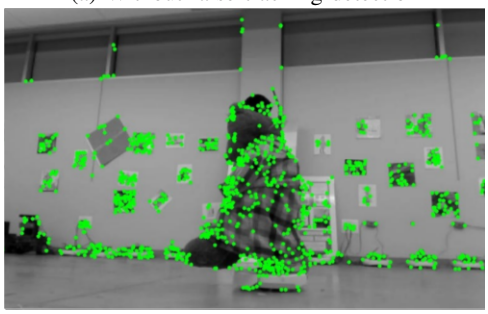
To handle dynamic scenes, map points are classified at every frame. All the map points are labeled as 'static' when they are first triangulated. At each frame, we check the reprojection error of the visible static map points by projecting them to the current and previous frames. A point will have a large reprojection error for two reasons: the point is dynamic or the point is generated from false feature matches. We make use of the observation that the dynamic points should be at the same position for different cameras at the same time. The 3D points are re-triangulated using the feature matches in different cameras at the same time. The points with small reprojection errors in individual camera are labeled as 'dynamic' while the others are 'false' points and should be discarded. The next three subsections cover modifications to make CoSLAM suitable for real-time operation.

1) Time Synchronization and System Initialization: In practice, robots send compressed images back with timestamps to the ground station over a wireless network. Time synchronization between frames is needed for dynamic points

Fig. 2: False feature tracks can be caused by the occlusion of the target. (a) The original feature location. (b) The feature starts drifting when the dynamic object enters the local window. (c) Since the dynamic object moves slowly, the change over two consecutive frames is small enough to pass the threshold for tracking. (d) The feature point will be tracked to a slightly different location, which snaps to the object's boundary.



(a) Without false tracking detection



(b) With false tracking detection

Fig. 3: Tracking results with and without false tracking detection. Falsely tracked features are concentrated on the target's boundary (blue rectangle) so that no points can be detected in the background (red rectangle).

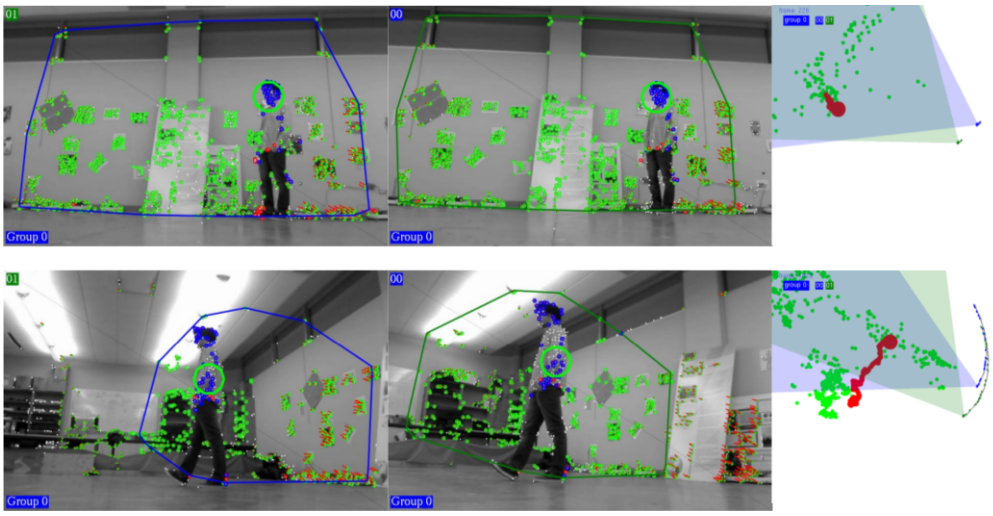


Fig. 4: Tracking results of two robots following a walking person at frame 226 (first row) and at frame 1025 (second row). Dynamic points (blue), static points (green), camera overlap region (polygons) and 3D map (right) are displayed. The estimated 3D position of the target is projected to both image planes as a green circle. The 3D target trajectory is drawn in red.

detection. *Network Time Protocol (NTP)* is used to synchronize the robot's computers and the ground station. The images are transmitted using *Robot Operating System (ROS)*. A ROS package called *message_filters*[1] synchronizes the images according to their timestamp.

We need to solve two issues before we can use camera poses obtained from CoSLAM to control the robots. First, the 3D map from CoSLAM has scale ambiguity. Second, the rigid transformation ${}^{\mathrm{C}}T_{\mathrm{W}}$ needs to be known between the camera frame F_{C} and the world frame F_{W} for all robots. We solve the two problems simultaneously by introducing AR tags to the initialization stage. Each AR tag is assigned 3D coordinates in F_{W}. We obtain the positions of the AR tags in the images using a ROS package called *ar_track_alvar*[2]. The 3D coordinates of the AR tags in F_{C} are triangulated using the initial estimated camera poses. The scaling factor s and the rigid transformation ${}^{\mathrm{C}}T_{\mathrm{W}}$ are solved by registering the 3D coordinates of the AR tags in F_{C} and F_{W}. The tags are removed after the system is successfully initialized. The camera poses are transformed into F_{w} before being sent to the controller. Note, that since scale is not corrected again once the system is running, it will degrade over time.

2) False Feature Tracking Detection: In our system, FAST feature points [11] are tracked using a sparse iterative version of the Lucas-Kanade optical flow algorithm [3]. Sufficient feature tracks should be maintained for accurate pose estimation and dynamic object detection. An OpenCL-accelerated KLT tracker in *OpenCV* is used to track around 1000 points within 5 ms for every frame.

During experiments, we observed that the static points in the background tend to be falsely tracked to the boundary of the moving object when the points are about to be occluded. Comparing only two consecutive frames, the KLT tracker tracks the feature point by searching within a local window. Since the target moves in slowly, the change over two consecutive frames is small enough to pass the threshold for tracking. As the original point is gradually occluded by the target, the tracked point starts drifting to a slightly different location, which will snap to the object's boundary. This is illustrated in Figure 2.

False feature tracks due to occlusion are a fatal issue for the original system since most of the background points would eventually be falsely tracked to the target's boundary. To address this problem, we have to re-check the feature's descriptor over a longer time period. We extract and compare the BRIEF feature descriptor [4] for the tracked feature points every 10 frames. A false feature track will be detected and discarded by computing the Hamming distance between the current descriptor and the previous one. The Hamming distance is large if the local window of the original feature position is partially occluded by the moving object. The comparison between the tracking results with and without the false feature tracking detection is shown in Figure 3.

3) Target Position Estimation: As feature points on the dynamic object are difficult to track, the distribution of dynamic points on the target changes over time. Moreover, point location inaccuracies are caused by motion blur and the rolling shutter effect during camera movement. These factors lead to noisy 3D reconstruction of dynamic points.

To obtain a stable position estimation of the target, we adopt a simple and effective approach. Since most of the dynamic points should be distributed in a constrained space of similar size to the target, we discard the points that are further than the expected distance travelled from the last observed target location. The geometric median of the remaining dynamic points is used to estimate the target's position. We further

[1] http://wiki.ros.org/message_filters
[2] http://wiki.ros.org/ar_track_alvar

smooth out the target's trajectory with a low-pass filter. The tracking results of a walking person are presented in Figure 4.

B. Controller

We implement a robot controller sympathetic to the requirements of CoSLAM for following a moving target; specifically, keeping the target in view of each camera and maintaining overlap between neighbouring cameras. By fulfilling these two requirements the robots are able to detect and follow a moving target with CoSLAM. Furthermore, we would like to ensure that the target does not occupy too much, or too little, of the camera view. In other words, there is a desired distance to keep from the target based on the target's size.

1) Orbiting: An orbiting controller similar to [10] allows our non-holonomic robots adapt easily to a variety of motions. The robots are able to keep up with a moving target assuming they move a magnitude faster than the target. As the optical axis of the camera is perpendicular to the robot heading, the goal is to drive perpendicular to the vector between robot and target to ensure that the target is centered in the camera view. In addition, the smoother rotations produced by orbiting are preferable to CoSLAM as opposed to sharp or erratic motions.

Given poses of the robot and target from the base station a desired global yaw angle θ_{D} for the robot is computed using Equation 1. In the 2D global polar coordinate system shown in Figure 5, θ_{T} is the polar angle of the vector $\mathbf{V}_{\mathrm{T}}$ from robot to target and θ_{O} is the polar angle of $\mathbf{V}_{\mathrm{O}}$ which is perpendicular to $\mathbf{V}_{\mathrm{T}}$. ρ is a weight between zero and one computed as a function of distance to target in Equation 2. We denote the distance between the target and image plane by d_{T}. As d_{T} increases, the robot needs to move closer to the target. The desired distance to target $\hat{d}_{\mathrm{T}}$ is determined a priori based on the desired target height as it appears in the image h_{D} (pixels). $\hat{d}_{\mathrm{T}}$ can be computed using h_{D}, focal length f of the camera, and the estimated physical height H_{T} of the target (See Equation 3). In practice the target's height may be fixed if a prior is available, or estimated from the 3D pose given by CoSLAM.

$$\theta_{\mathrm{D}} = \rho \cdot \theta_{\mathrm{T}} + (1 - \rho) \cdot \theta_{\mathrm{O}} \tag{1}$$

$$\rho = min(1, max(0, -(\hat{d}_{\mathrm{T}}/d_{\mathrm{T}}) + 1)) \tag{2}$$

$$\hat{d}_{\mathrm{T}} = \frac{f \cdot H_{\mathrm{T}}}{h_{\mathrm{D}}} \tag{3}$$

The combination of angles used to produce θ_{D} lets the robot adjust its distance to the target dynamically. Figure 5 provides an overview of the 2D geometry between robot and target. A PID controller is used to throttle the robot's rotational velocity in order to approach θ_{D}. This PID controller with a constant velocity produces trajectories similar to Figure 6.

2) Keeping Target in View: It is possible that rotational motions produced by the orbiting controller cause the target to be outside field-of-view (FOV) θ_{FOV} of the camera. To prevent losing sight of the target an upper bound $\theta_{\mathrm{UB}} = \theta_{\mathrm{O}} + \theta_{\mathrm{FOV}}/2$ and lower bound $\theta_{\mathrm{LB}} = \theta_{\mathrm{O}} - \theta_{\mathrm{FOV}}/2$ are computed for θ_{D}. A check is done to see if the vector produced by θ_{D} is to the left of its upper bound or to the right of its lower bound and clamped accordingly. This constraint is particularly useful if the target is first detected too far away and the robots need to close the gap without losing sight. In the event that the target cannot be detected, the robots will use the target's last known position in an attempt to recover.

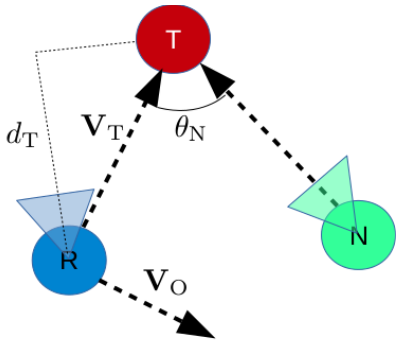


Fig. 5: A robot (blue) and its neighbor (green) are orbiting a moving target (red).

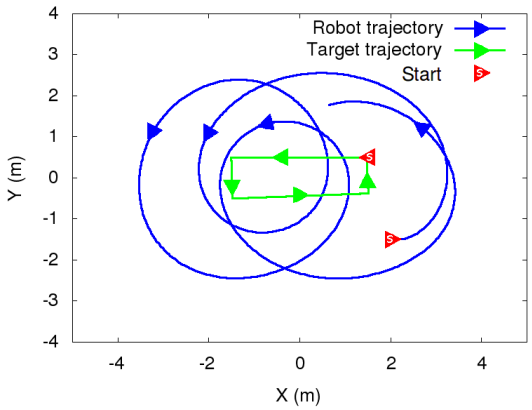


Fig. 6: The trajectory of a single robot orbiting a moving target in simulation.

3) Maintaining Overlap: So far we have only considered a single robot orbiting a target, but in order to detect a dynamic target CoSLAM requires camera overlap. Therefore, we apply a final constraint that will throttle the robot's forward velocity given the position of its nearest neighbour. The idea is to have each robot maintain a distance with another robot in the orbit. A neighbour robot can be chosen a priori or dynamically simply by choosing the closest robot.

A desired distance $\hat{d}_{\mathrm{N}}$ to a robot's neighbour is computed with Equation 4. $\hat{\theta}_{\mathrm{N}}$ is the desired angle between a robot and its neighbour in the orbit. As $\hat{\theta}_{\mathrm{N}}$ becomes smaller more overlap is created between cameras. If the actual distance to a robot's neighbour is less than $\hat{d}_{\mathrm{N}}$, then the robot is too close and risks collision. Likewise, if the distance is greater than $\hat{d}_{\mathrm{N}}$, then the robot is too far away, reducing overlap. Next, the bearing to a neighbour is used to determine if the robot is in

front or behind. Each robot follows a simple set of rules listed in Table I to set its velocity. A PID-controller helps smooth changes in velocity.

$$\hat{d_{\mathrm{N}}} = 2\hat{d_{\mathrm{T}}} \cdot sin(\hat{\theta_{\mathrm{N}}}/2) \tag{4}$$

TABLE I: Rules for adjusting robot velocity.

Neighbour distance	Bearing to neighbour	Result
Too close	Behind	Decrease velocity
Too close	Front	Increase velocity
Too far	Behind	Increase velocity
Too far	Front	Decrease velocity

IV. Experiments

The prototype system was tested in two different environments: a visual-feature-rich lab (Figure 1) and a sparsely featured boardroom. The data described below is from a representative run in the lab environment. The accompanying video[3] shows a demonstration in the boardroom environment.

A. Apparatus

The robots in our experiments were the iRobot Create 1 with an onboard Odroid U3 (cellphone-class) computer to compress and transmit camera frames and a Gumstix computer for robot control. The base station was a laptop with an Intel Core i7 (2.5 GHz) processor and NVIDIA GeForce GTX 850M graphics card. CoSLAM ran on the laptop posting pose information to a Redis server. The target was a stuffed teddy bear attached to a non-cooperating robot standing 0.86 metres tall.

B. Method

In each experiment a target travelled a pre-planned rectangular trajectory until it covered a set distance. Two or three robots were given the task of tracking the target. After CoSLAM initialization, the target started driving into view of the robot's cameras. Once detected, the controller was engaged and the robots began their encirclement. We let the system run until the target completed a full traversal of a three metre by one metre rectangle.

The target drove at a speed of $0.06m/s$, while the robots drove at around $0.3m/s$. Slow speeds were chosen to avoid motion blur and rolling shutter artifacts. The target height of $0.86m$ was set a priori and desired height ratio in the view was 0.6. This means that the robots should encircle with a radius of around $1.92m$. Neighbours were manually assigned to maintain 30 degrees between them in the circle.

C. Results

We consider three errors: *yaw error*, *ratio error* and *neighbour error*. Yaw error is defined as the difference between θ_C and robots yaw. Ratio error is the difference between ratio of the target's height in the image and the desired ratio. Neighbour error is the difference between θ_N and $\hat{\theta_N}$. These three errors reflect the constraints outlined in III-B.

[3]https://www.youtube.com/watch?v=uYQ_Bmv3LkI

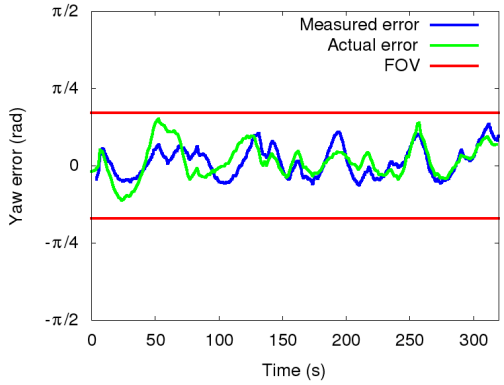


Fig. 7: Yaw error over time (blue) compared to error obtained from ground truth (green). Although error fluctuates target never leaves field-of-view (red).

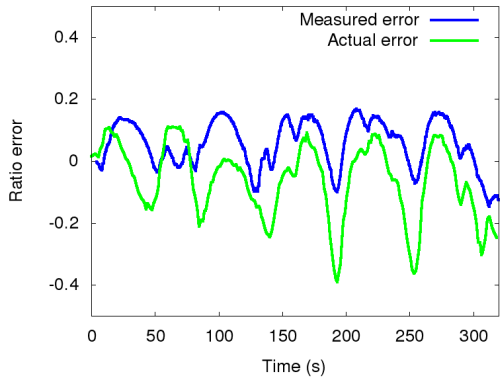


Fig. 8: Ratio error (blue) compared with error obtained from ground truth (green). The ratio measured is the target's height in the camera view.

We want to show that the errors measured by the system actually correspond to the errors in the real world. To support this assertion, we examined the video recorded by the cameras on the robots themselves. By manually annotating the videos we gained a ground truth measurement for the yaw error and ratio error over time. We were unable to obtain ground truth measurements for the neighbour error.

The representative results shown are from a run performed in the lab environment with two robots. Yaw error for one of the robots can be seen in Figure 7. The correlation between the measured and actual error shows that CoSLAM provides reasonable localization that is required to keep the target in view. Discrepancies are caused by changes in the distribution of features on the target due to changes in viewing angle. Different distributions move the median around the target's actual centre.

A similar correlation can be observed in Figure 8 for ratio error, again showing reasonable localization from CoSLAM. However, the difference between the measured and actual error increases over time caused by scale drift in CoSLAM's measurements. Robots will continue to increase their radius

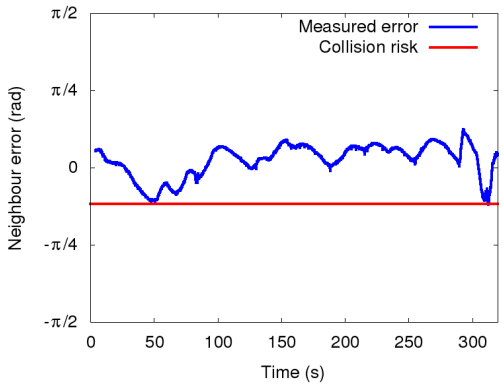


Fig. 9: Neighbour error (blue) over time. Lower error increases overlap with neighbours camera view. The red line indicates that the robots are too close and risk colliding.

from the target increasing the risk of collision with the environment's boundary. Correcting scale drift in the future will allow for a more stable system.

It is difficult to infer how accurately the measured neighbour error reflects their actual error without a ground truth. However, the neighbour error in Figure 9 accurately reflects the fact that the robots never collided.

We find that the system is able to successfully track the moving target, keeping it in view of at least two cameras at all times. CoSLAM provides the necessary pose information to direct the controllers and the robots move to allow CoSLAM to continue target tracking. The robots orbit the target as it travels, so it is observed from all sides over time. This is the behaviour we seek for our system to eventually capture 3D models of the target in real time.

V. CONCLUSION

This is an initial demonstration of the feasibility of cooperative visual-SLAM-only control for robot teams to scan a moving target. The system currently captures 3D point clouds of the target, but we have not yet investigated their quality or scope for building solid, time-varying target models - this is a next step for this work. Our future work will also include larger robot teams, where managing the wireless bandwidth is a practical problem, and using faster robots with more degrees of freedom of motion, including UAVs. Also of interest is creating a fully distributed version of CoSLAM, that could be tolerant of intermittent communications or individual vehicle failures.

REFERENCES

[1] Markus Achtelik, Yorick Brunet, M Chli, S Chatzichristofis, Jean-dominique Decotignie, Klaus-michael Doth, F Fraundorfer, L Kneip, D Gurdan, L Heng, et al. Sfly: Swarm of micro flying robots. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2649–2650. IEEE, 2012.

[2] Miguel Aranda, Gonzalo López-Nicolás, Carlos Sagüés, and Michael M. Zavlanos. Three-dimensional multirobot formation control for target enclosing. In *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*, pages 357–362. IEEE, 2014.

[3] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5, 2001.

[4] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer, 2010.

[5] Chun-Hua Chang, Shao-Chen Wang, and Chieh-Chih Wang. Vision-based cooperative simultaneous localization and tracking. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5191–5197. IEEE, 2011.

[6] AnhDuc Dang and Joachim Horn. Formation control of leader-following uavs to track a moving target in a dynamic environment. *Journal of Automation and Control Engineering Vol*, 3(1), 2015.

[7] Vikas Dhiman, Julian Ryde, and Jason J Corso. Mutual localization: Two camera relative 6-dof pose estimation from reciprocal fiducial observation. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1347–1354. IEEE, 2013.

[8] A Dias, J Almeida, P Lima, and E Silva. Uncertainty based multi-robot cooperative triangulation. In *RoboCup Symposium Proceedings, Brasil. LNCS (LNAI). Springer*, 2014.

[9] Christian Forster, Simon Lynen, Laurent Kneip, and Davide Scaramuzza. Collaborative monocular slam with multiple micro aerial vehicles. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3962–3970. IEEE, 2013.

[10] Antonio Franchi, Paolo Stegagno, Maurizio Di Rocco, and Giuseppe Oriolo. Distributed target localization and encirclement with a multi-robot system. In *7th IFAC Symposium on Intelligent Autonomous Vehicles*, pages 151–156, 2010.

[11] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[12] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[13] Peng Yang, Randy A Freeman, and Kevin M Lynch. Distributed cooperative active sensing using consensus filters. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 405–410. IEEE, 2007.

[14] Danping Zou and Ping Tan. Coslam: Collaborative visual slam in dynamic environments. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(2):354–366, 2013.