Leader-Follower Formations on Real Terrestrial Robots

Alexandru Solot
University of Cambridge
ams288@cam.ac.uk

Andrea Ferlini
University of Cambridge
af679@cam.ac.uk

ABSTRACT

Controlling robot formations represents one of the biggest open challenges for the Robotics Community. Its importance is remarked in a wide range of applications from truck platooning to autonomous robot-based parcel delivery. In this paper we propose a lightweight decentralized solution for controlling a queue shaped leader-follower turtlebot formation. In order to test our implementation, we perform tests both in simulation and on real turtlebots[1]. The complexity of the challenge we are tackling further increases due the fact that the robots we use to evaluate our work are non-holonomic, meaning they have limited controllable degrees of freedom.

CCS CONCEPTS

• **Computer systems organization** → *Robotic control*; • **Embedded and cyber-physical systems** → *Robotic autonomy*; *Embedded software*;

KEYWORDS

robot formation, decentralized robot formation, leader-follower formation, lidar tracking

ACM Reference Format:
Alexandru Solot and Andrea Ferlini. 2019. Leader-Follower Formations on Real Terrestrial Robots. In *MAGESys '19: ACM SIGCOMM 2019 Workshop on Mobile Air Ground Edge Computing, Systems, Networks, and Applications, August 19, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341568.3342107

1 INTRODUCTION

Formation control represents nowadays one of the most important research areas in mobile robotics. The gain in popularity of self-driving vehicles and their increasing demand on the market, the ubiquity of unmanned aerial vehicles (UAVs), and the necessity of efficient fleets of autonomous underwater vehicles (AUVs) has broadened the spectrum of formation-control scenarios.

To solve the formation control problem, three main approaches have been outlined by literature: behavioural based, virtual structure based, and leader follower based approaches. In the leader follower case, a designated leader moves along a trajectory while the other robots are supposed to follow it preserving the formation and a specified distance.

One of the reference papers regarding leader-follower formation addresses the control of non-holonomic robots and was published by Consolini et al.[4]. In this paper concepts such as formation stability were formalized. Other techniques such as using complex laplacians [12], receeding-horizon schemes [1], or neuro-dynamic approaches [13] were proposed in order to solve the problem.

However, tackling obstacle avoidance while still moving in formation remains an open question. Existing work in this field could be divided in two categories: learning-based approaches, and model-based approaches. Learning based approaches are trying to leverage the way humans take decisions by using machine learning, while the model-based solutions rely on geometric rules and physics considerations. One of the most representative model-based approach is the velocity obstacles (VO) model [6].

Recent work, published by Karamouzas et al. in 2015 [11] is leveraging this approach, proposing the Formation Velocity Obstacles, which allow anticipatory collision avoidance and prioritization of formation reshaping. More recent papers published in 2018 are proposing hybrid combination between RRT* and a generalized version of VO in order to perform real-time non-holonoic robot navigation in dynamic environments [2] with potential extension to formations.

A leader-follower formation consists of a designated leader which moves along a trajectory, and follower robots which are supposed to follow the leader preserving the formation and a specified distance. For non-holonomic mobile robots formations, the task of following the leader becomes extremely difficult given that robot inputs are forced to satisfy suitable constraints. This restricts the set of possible paths and implicitly the formation's shape, configuration, and robustness.

In this work we simplify the generic task previously described and we focus on maintaining a leader-follower queue formation using a decentralized approach. We aim to control a number of 3 non-holonomic robots, each taking decisions based only on the information provided by the on-board lidars, and acting accordingly in order to follow either a human or another robot. Moreover, we extend the approach to demonstrate robustness to dynamic obstacles that might disrupt the formation.

To wrap up, the main contributions of our work are:

- We implemented two detection variants and two tracking variants.
- We implemented two variant combinations of detection with their respective tracking modules.
- We successfully tested and evaluated our work on real robots in a real world scenario.

2 SYSTEM DESIGN

As part of this work we have designed a flexible navigation solution wrapped into a ROS package named `turtlebot3_formation`. The navigation system controls each turtlebot entity independently

[1]Code could be found at https://github.com/ASolot/turtlebot3_formation

MAGESys '19, August 19, 2019, Beijing, China

ACM ISBN 978-1-4503-6879-7/19/08…$15.00
https://doi.org/10.1145/3341568.3342107

from others, and could be scaled to an arbitrary number of robots thanks to ROS communication topics being clustered under robot-specific name-spaces.

The system is able to work with plug-and-play detectors and controllers, with the aim of supporting various input sensors, and different formation control strategies. For the scope of the current project we have developed two interchangeable detectors and two interchangeable controllers. The implementation was done in Python using the RosPy client library [3] for the Kinetic ROS target distribution. The schematic representation of the control structure is depicted in Figure 1.

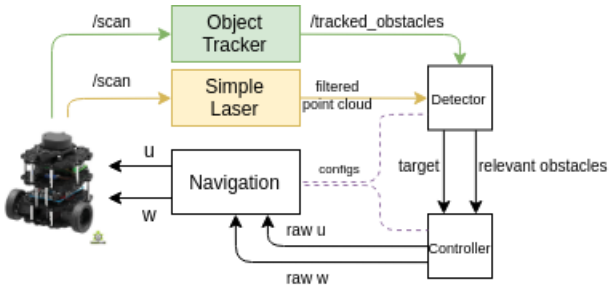

Fig. 1: Our proposed modular detector - controller structure

The first detector-controller pair (A1-B1) leverages a lightweight custom-built object tracker and a PID based go-to-goal controller, aiming to provide a simple and fast method for controlling the formation. The second approach (A2-B2) uses an off the shelf obstacle detector ROS module by Mateusz Przybyla [14], and a Velocity Obstacles based controller.

Given our problem definition, no formation supervisor was implemented. Moreover, path planning algorithms were excluded due to their computation time overheads, that would have significantly slow down the overall system. Thus, the queue formation results as an emergent property.

2.1 Detection

We define the detection problem as identifying and computing the coordinates of objects such as legs, turtlebots, or other obstacles by using the 2D lidar point cloud published by each robot.

2.1.1 ***Clustering-based detection.*** In order to solve the problem of detecting legs and turtlebots based on laser scan data, we propose a lightweight method that uses Density-based spatial clustering (DBSCAN) [5].

The first step is to reduce computational complexity which is correlated with the number of points to be clustered. We firstly reduce the field-of-view to 140°, and then we aggregate the original data into 72 measurements having a resolution of 2° and a field-of view cone of 1° per measurement. The result of this operation is depicted in Figure 2.

The second step is to perform clustering and filter the clusters based on parameters such as total number of points, maximum distance between the points in the cluster, and distance from the robot to the cluster. As it could be seen in Figure 3, this operation filters down noise and other unwanted obstacles, the resulting

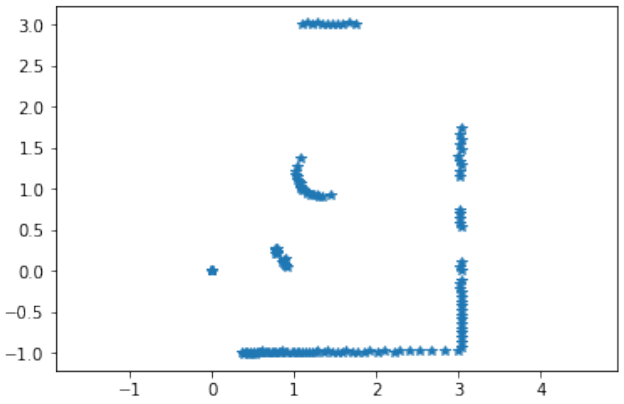

Fig. 2: The input laser points for the DBSCAN detector

clusters being considered as potential followable objects. Finally, the coordinates of the closest cluster centroid are chosen as the target object coordinates.

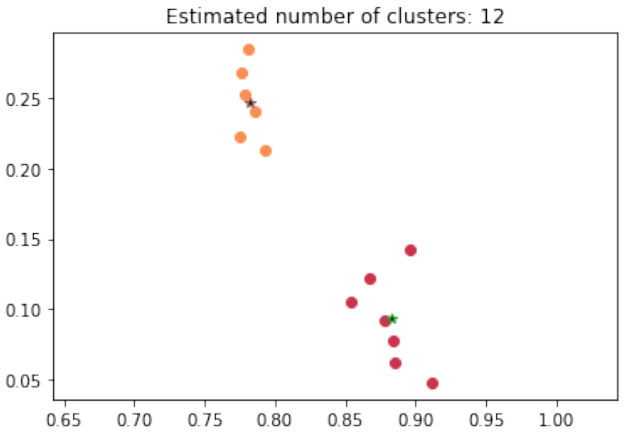

Fig. 3: The result of clustering based feet detection

2.1.2 ***Dynamic detection and tracking.*** Given the simplicity and the limitations of our proposed method, we have considered a second option for performing obstacle detection and target identification. The method was proposed by Mateusz Przybyla in [14] which is openly available on github[2] and makes use of a Kalman filter for object tracking and object speed detection. The algorithms run using separate ROS nodes, which publish the object coordinates, speeds, and sizes using a custom defined message type. The results are depicted in Figure 4.

The target is identified using another implemented detector which makes use of the published obstacle coordinates, their speed, and their displacement between two successive detection. As a result, tracking is performed consistently over time and with a greater robustness to noise.

[2]https://github.com/tysik/obstacle_detector

Fig. 4: The result of the advanced detection algorithm

2.2 Tracking

Given a target point within the robot's field of view, we define the tracking problem as ensuring that the robot follows the point while keeping a constant distance from it.

2.2.1 ***Go-to-Goal PID.*** One of the most computationally effective method for solving the point-tracking problem represents implementing a go-to-goal controller. In the case of a turtlebot, the controllable parameters over time are the linear speed u and angular velocity w. Thus, the tracking situation could be reduced to the one depicted in Figure 5, where our go-to-goal controller aims to minimize the distance error and the heading error between the robot and the target.

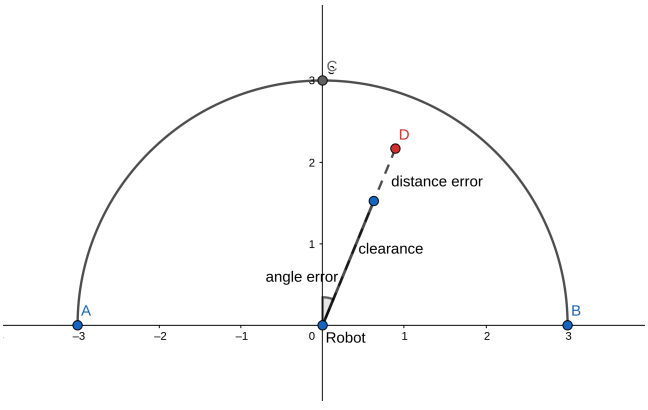

Fig. 5: Go-to-goal problem formulation for a turtlebot

In practice, this controller could be implemented following the Equations 1 and 2 as two independent PID controllers. In certain situation, an artificial relation between the two parameters e_u and e_w could be created by formulating the distance error control as a horizontal displacement control problem: $e_{u_{new}} = e_u cos(e_w)$. This ensures smoother reactions to sideways movement of targets.

$$u[k] = K_{p_u} e_u[k] + K_{i_u} \sum_{t=0}^{k-1} e_u[t] + K_{d_u}(e_u[k] - e_u[k-1]) \quad (1)$$

$$w[k] = K_{p_w} e_w[k] + K_{i_w} \sum_{t=0}^{k-1} e_w[t] + K_{d_w}(e_w[k] - e_w[k-1]) \quad (2)$$

2.2.2 ***Velocity Obstacles.*** For an environment cluttered with moving obstacles, the approach previously presented would not perform reasonably as it ignores all other objects apart from the target. Thus, to solve this problem, we propose to use a modified version of the Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation proposed by Van den Berg et al. [15] and available on github [8].

The Velocity Obstacles approach, as defined in Figure 6 works based on the following principle: during a planning cycle, the agent picks a velocity that is not intersecting any of the velocity obstacles induced by the moving obstacles. The velocity chosen guarantees safe navigation towards the goal only if the chosen velocity is directed to the agent's goal position. In the Reciprocal Velocity Obstacles case, the idea behind is to choose a new velocity that is the average of its current velocity and a velocity that lies outside the other agent's velocity obstacle. Thus, this new method provides a simple approach to safely and smoothly navigate multiple agents among each other without explicit communication between them.

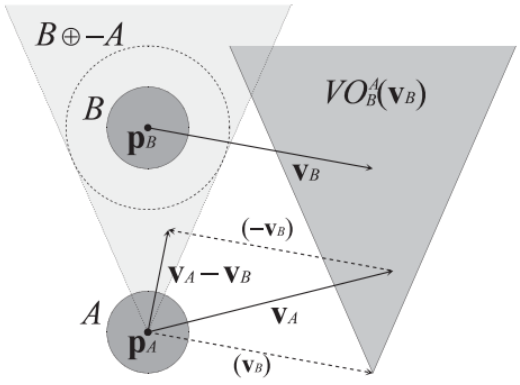

Fig. 6: The Velocity Obstacle of a disc-shaped obstacle B to a disc-shaped agent A

We configure the available implementation to use the Velocity Obstacles approach while solving the multi-agent problem, as moving obstacles would not adjust their speed in order to avoid a potential collision. Moreover, given that our scenario would not imply symmetrical collision-avoidance strategies, we are not going

to be influenced by the main downside of plain Velocity Obstacles which is the generation of oscillatory motions leading to slow convergence.

Using the A2 detector, we obtain all the coordinates and speeds of the obstacles, and feed them into the RVO algorithm. All the values are expressed in the coordinate system of the moving robot, which is considered to be continuously located in the origin. The resulting feasible speed for our robot, which is expressed as a raw speed vector, is then transformed using feedback-liberalization (Equations 3 and 4) into a valid u and w command. In the mentioned equations, ϵ denotes the distance between the robot and the holonomic point for which the raw speeds were computed, and yaw represents in this case the slope of the line determining the speed vector's direction.

$$u = v_x cos(yaw) + v_y sin(yaw) \tag{3}$$

$$\omega = \epsilon^{-1}(-v_x sin(yaw) + v_y cos(yaw) \tag{4}$$

2.3 Map merging

During the testing phase, we realized that unifying the visualization of the obstacles detected by all the robots in the formation was necessary. Thus, we have decided to use the `multirobot_map_merge` [10] ROS package to create a merge between the maps generated independently by our robots. To maintain flexibility between simulation and real-world deployment, we have opted for map merging without known initial position. The results are depicted in Figure 4.

3 EVALUATION

3.1 Methodology

We perform a set of experiments for each proposed detector-controller pair. We aim to (1) test the navigation capabilities of our formation in an environment with static obstacles, and (2) test formation robustness to dynamic obstacles.

All of the experiments were firstly performed in simulation environment. Due to physical constraints such as latency issues that were severely limiting the performance, only the navigation test using the A1-B1 pair was performed on real robots.

3.2 Launching an experiment

The code is structured as a ROS package. Detailed instructions on how to install and configure the package could be found in the Readme file from our github repository. To ease experiment deployment, we have created a bash script called `deploy.sh` which is responsible of triggering all the necessary software.

3.2.1 ***Simulation experiment.*** In order to run a simulation experiment, open a terminal in the root folder of the `turtlebot3_formation` package, and run `./deploy.sh`. The bash script will automatically launch the Gazebo environment, the RViz visualizer, and the python navigation scripts for the three turtlebots. To edit navigation related parameters, open the `deploy.sh` and edit the arguments with which the `ros/navigation.py` python script is called.

3.2.2 ***Real-world experiment.*** In order to run a real-world experiment, follow the same steps as above, and run `./deploy.sh real`. The bash script will automatically launch the RViz environment, the ROS Master, the clock synchronization script, the navigation control scripts, it will configure the network addresses and will launch the ssh connections to the robots. However, due to security reasons, the password and the automatically generated robot-specific commands should be copy-pasted manually in the robot terminals.

3.3 Results

3.3.1 ***Navigation in formation.***

A1-B1 Configuration. For this specific configuration, we evaluated how the K_p, K_i, and K_d parameters of the two PID controllers impact the overall navigation performance.

One instant remark is that for the distance PID, the distance error could never become negative. Thus, the integral member of the equation would grow unbounded, and the motors would get stuck with a saturated command. As a consequence, we do not allow setting any $K_{i_u} \neq 0$.

In the first run of our experiments, we begin with $K_{p_u} = 1.2$ and $K_{p_w} = 0.5$, and all other constants set to 0. The PID responses of the demo presented in the video could be seen in the Figures 7 and 8.

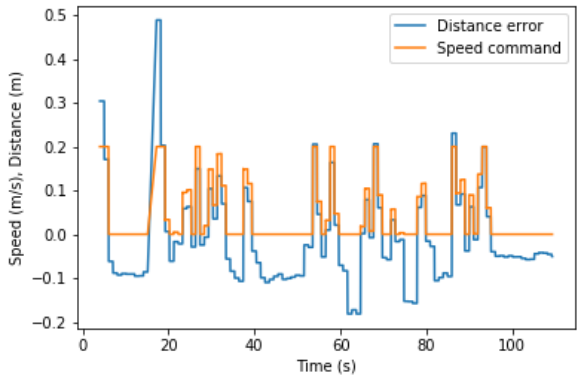

Fig. 7: P controller response for $K_{p_u} = 1.2$

For high values of the K_p coefficient we observe in Figure 7 that the controller saturates fast. This behaviour is undesired as it results in shocks that could potentially damage the motors. Even though the saturation problem is less visible in the angle case (Figure 8), we decided to lower both proportional coefficients.

Apart from lowering the K_p coefficients to $K_{p_u} = 0.6$ and $K_{p_w} = 0.1$, we have also introduced an integral coefficient for the angle controller, in order to solve the stationary error problem that could be seen in Figure 8. The results are depicted in Figures 9 and 10.

We have neglected to introduce a derivative component as part of our controllers due to noise related concerns. As the derivative component has the role of reacting to the variation of the error, this could lead to spikes of the output command in the real world environment which is dominated by noisy measurements.

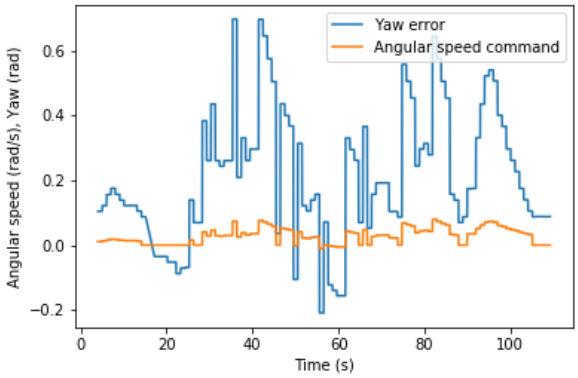

Fig. 8: P controller response for $K_{p_w} = 0.5$

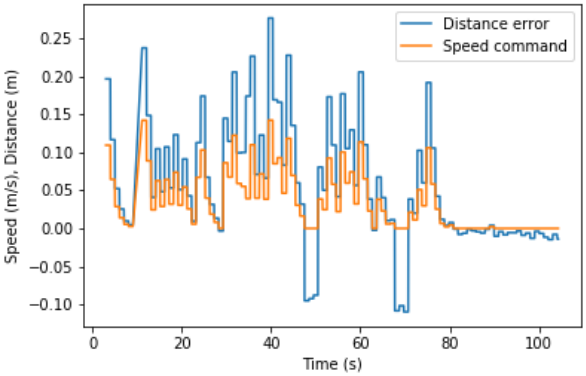

Fig. 9: P controller response for $K_{p_u} = 0.6$

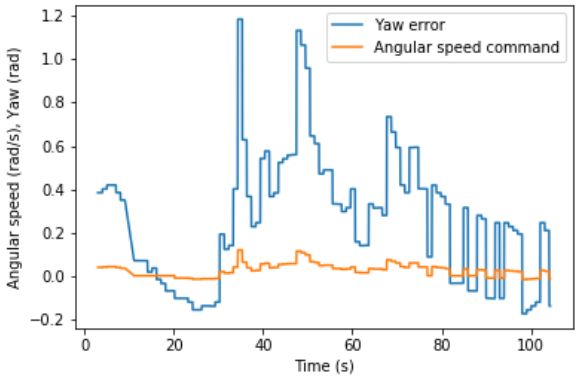

Fig. 10: PI controller response for $K_{p_w} = 0.1$ and $K_{i_w} = 0.01$

In simulation, using the previously tuned controller, we managed to control effortlessly the formation around the obstacle track. The results could be seen in Figure 11.

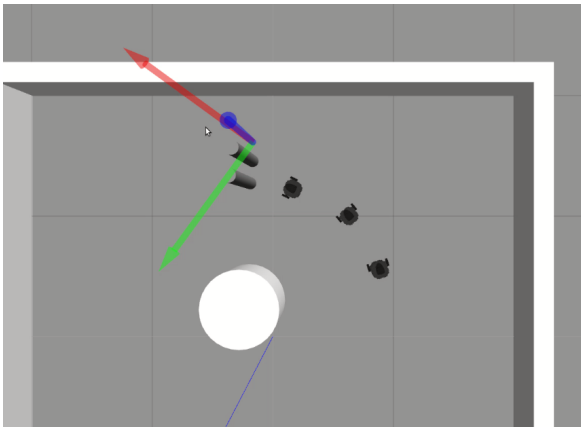

Fig. 11: Example of formation control around the obstacle course using the A1-B1 configuration

However, in the real world deployment of this scenario we observed that the already tuned parameters were no longer suitable for the task. Thus we needed to re-tune the controllers in order to obtain an acceptable behaviour. We managed to run the control loop at 5 Hz and the target detection loop at 3.3 Hz for the three turtlebots. The results could be seen in the attached video.

A2-B2 Configuration. In the absence of obstacles, the Velocity Obstacles based controller moves the robots using the maximum allowable speed. During the simulations we observed steep transitions from 0 to maximum speed each time the target was moving rapidly. This aspect represents a severe inconvenience for a real-world deployment and needs to be addressed by implementing a smoothing function for start-stop. However, the formation is kept with better performance than in the previous configuration, due to a fast response to drastic target position changes.

3.3.2 ***Robustness to dynamic obstacles.*** In this test, we assessed how the formation is influenced by dynamic obstacles.

A1-B1 Configuration. Given neither the detector nor the controller are taking into consideration moving obstacles and their speeds, this configuration shows little robustness to dynamic obstacles. Usually obstacles would trigger a halt, and the robots would break the formation, as it could be seen in Figure 12

A2-B2 Configuration. Even though promising, the method showed little robustness to dynamic obstacles. Due to the formation nature and the target detection system, after adjusting the trajectory in order to avoid collision, the robot will mistakenly identify the obstacle as the new target. Thus, the robustness could be tested only given specific scenarios in which the robots would not be kidnapped by the obstacle.

4 FUTURE WORK

Using lightweight local path planning techniques such as Optimal Trajectory Generation using Frenet Frames [16], or the Dynamic window approach [7] might offer a more robust alternative in the context of target positioning failure. However, due to the fact that we focused our effort on designing a decentralized control scheme,

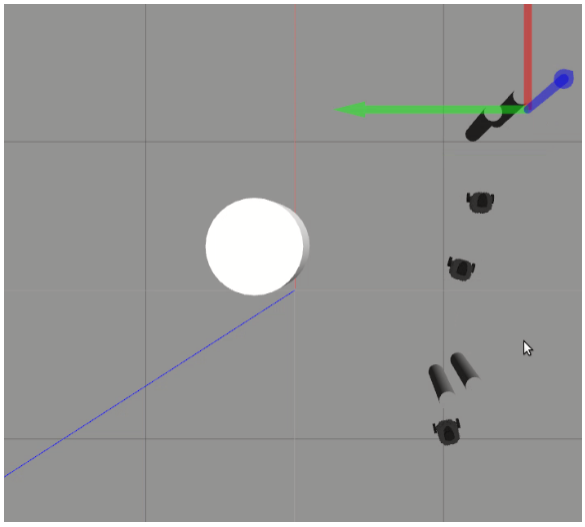

Fig. 12: Formation disruption by a dynamic obstacle for the A1-B1 configuration

we decided to leave the optimization of the path planning as a future work. A further improvement could consist in the deployment of parts of the computational workload directly on the turtlebots, in order to remove the latency of the link between the computational nodes and the robots. Moreover, our evaluation was carried out in a controlled laboratory environment. Thus, no real-world network intermittent behaviour is accounted. In order to address this open issue, we plan to use our delay-tolerant robot-communication protocol. This could be a turning point for one of the future applications we envision, where instead of following human beings, ground vehicles follow aerial robots and/or emergency vehicles in critical situations as rescue applications.

5 FINAL REMARKS

As shown in the evaluation part, we have successfully managed to implement a lightweight decentralized solution for controlling a queue shaped leader-follower turtlebot formation. However, this was possible only in the case of the A1-B1 detector-controller pair and under the assumption that no dynamic obstacles are present in the environment.

The inaccuracies of the proposed detectors were the main contributors to the low performance of the velocity obstacles method. Moreover, given we are operating in a decentralized manner, the lack of a method that would predict the target's next location based on previous poses in the case of a laser obstruction has consequences such as robot kidnapping by moving obstacles.

Differences between the simulation and the real world experiments were also noticed. Firstly, all the simulation-tuned controller parameters had to be re-tuned for the real-world approach. Secondly, due to the fact that the computational nodes were run on a separate device, time overheads were introduced between the measurement events and the reaction events. The effect of this issue was clear: delays, slow and non-consistent feedback loops as we got farther away from the router.

Sensors posed another challenge for achieving good results. While lidars are very accurate on obstacle positions, they are less accurate on detecting their velocities. Thus, avoiding dynamic obstacles while maintaining the formation represents a complex problem. Novel approaches such as sensor fusion between lidars and radars were recently proposed by Hatem et al in [9] in order to overcome this issues.

Additional approaches to get reliable distance information such as using a Kinect generated point clouds were also considered. Even though promising results were obtained using both `ppl_detection`[3] and `cob_people_detection`[4] packages, we decided not to include a Kinect in our final solution due to size and power consumption constraints.

ACKNOWLEDGEMENTS

This work has been carried out during the course Mobile Robot Systems taught by Dr. Amanda Prorok at the University of Cambridge, UK[5].

REFERENCES

[1] J. Chen, D. Sun, J. Yang, and H. Chen. Leader-follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme. *The International Journal of Robotics Research*, 29(6):727–747, 2010.

[2] Y. Chen, M. Liu, and L. Wang. Rrt* combined with gvo for real-time nonholonomic robot navigation in dynamic environment. In *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 479–484. IEEE, 2018.

[3] K. Conley. rospy - python client library for ros. https://github.com/ros/ros_comm.git, 2019.

[4] L. Consolini, F. Morbidi, D. Prattichizzo, and M. Tosques. Leader–follower formation control of nonholonomic mobile robots with input constraints. *Automatica*, 44(5):1343–1349, 2008.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Density-based spatial clustering of applications with noise. In *Int. Conf. Knowledge Discovery and Data Mining*, volume 240, 1996.

[6] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[7] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[8] M. Guo. Python implementation of reciprocal velocity obstacle for multi-agent systems. https://github.com/MengGuo/RVO_Py_MAS, 2019.

[9] H. Hajri and M.-C. Rahal. Real time lidar and radar high-level fusion for obstacle detection and tracking with evaluation on a ground truth. *arXiv preprint arXiv:1807.11264*, 2018.

[10] J. Horner. Multirobot map merge. http://wiki.ros.org/multirobot_map_merge, 2019.

[11] I. Karamouzas and S. J. Guy. Prioritized group navigation with formation velocity obstacles. In *2015 IEEE International*

[3] http://wiki.ros.org/ppl_detection
[4] http://wiki.ros.org/cob_people_detection
[5] https://www.cl.cam.ac.uk/teaching/1819/MobRobot/

Conference on Robotics and Automation (ICRA), pages 5983–5989. IEEE, 2015.

[12] Z. Lin, W. Ding, G. Yan, C. Yu, and A. Giua. Leader–follower formation via complex laplacian. *Automatica*, 49(6):1900–1906, 2013.

[13] Z. Peng, G. Wen, A. Rahmani, and Y. Yu. Leader–follower formation control of nonholonomic mobile robots based on a bioinspired neurodynamic based approach. *Robotics and autonomous systems*, 61(9):988–996, 2013.

[14] M. Przybyła. Detection and tracking of 2d geometric obstacles from lrf data. In *2017 11th International Workshop on Robot Motion and Control (RoMoCo)*, pages 135–141. IEEE, 2017.

[15] J. Van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[16] M. Werling, J. Ziegler, S. Kammel, and S. Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE, 2010.