Ducklings Follow Leader

A multi-agent system approach using the Turtlebot2

Daniel Fernandes Gomes and Luis Costa
Informatics Engineering Department.
Faculty of Engineering of University of Porto, FEUP
Porto, Portugal
{up201306839, ei12008}@fe.up.pt

***Abstract* — This project aims to look into the implementation of a Multi-Agent System where autonomous robots follow each other to perform predetermined trajectories. The ROS framework and the Turtlebot2 are used and evaluation performed in Gazebo simulator.**

Robotics; simulation; computer vision; multi-agent; following; ROS; gazebo; opencv; pointcloud.

I. Introduction

As time goes by, with the development of new technologies, robots are getting more and more complex. Technologies such as Computer Vision, which allows a system to recognize other objects, and Multi-Agent systems, in which several autonomous agents interact with each other towards a common goal, are well known and often discussed subjects in the robotics community. In this paper, we present a way to combine both technologies in order to create a system of robots that detect and follow each other and a leader, throughout several predefined trajectories.

The implementation is made using the Robotic Operating System (ROS) [1] and the Turtlebot2 robot under the simulation environment provided by Gazebo.

II. Capabilities and Strategies

A. *Walking predetermined trajectories*

The simplest configuration of the system has only one robot, the leader, which must be able to travel along a predefined trajectory. Instead of implementing for a specific trajectory or use more complex modeling techniques, such as splines, our approach involves chaining two parameterizable primitives: lines and ellipses.

Lines have only one parameter: the size, or distance. While ellipses can be configured regarding its two perpendicular radius, its orientation (clockwise or the opposite) and the amplitude of travel. These primitives can then be executed in order only once or in loop, to perform more complex trajectories, such as the eight shape, using two ellipses with different orientations.

As the equations of each primitive are well known, this strategy can be implemented as follows: in each instant, calculate the next position, and the movement vector becomes the difference between the next and the current positions.

For the line this becomes

$$\mathbf{P}_t = \mathbf{S} + (t \,/\, \text{duration}) \times \text{distance} \times [\cos(\alpha), \sin(\alpha)]$$

where the duration is calculated as the velocity × distance and α is the initial yaw. And for the ellipse

$$\theta_t = O \times (t \,/\, \text{duration}) \times 2\pi.$$

$$\mathbf{P'}_t = [R_1\cos(\theta_t), R_2\sin(\theta_t)]$$

$$\mathbf{P}_t = \mathbf{C} + \mathit{rotate}(\mathbf{P'}_t, \ \alpha + \pi/2 - 2\pi)$$

where O is an binomial 1 or -1 concerning the orientation of travel, $\mathbf{P'}_t$ is the position in an ellipse centered on the origin without any rotation and $\mathbf{P}_t$ is the final position. **C** is the center of the trajectory.

Because of its limited turning rate, the robot will often deviate from the ideal trajectory and outpace the plan, when performing ellipse trajectories. Applying the initial strategy results in an exponential deviation from the desired path. To solve this, only rotation is, in this scenario, applied and the robot stays still until the angle between its orientation and the movement vector is lower than pi/2.

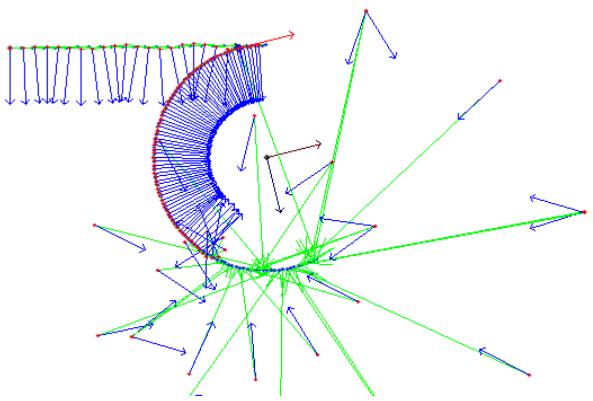
Figure 1. The robot outpaces the plan and deviates exponentially from the plan. Red dots are the robot sampled position, blue arrows the perpendicular to the orientation of the robot. Dots in blue are planned positions. Green arrows are the movement vector.

B. *Recognizing and following*

When multiple robots coexist, the first is the leader while the remaining are the followers. Each follower must be able to recognize the leader and move towards it.

The recognizing task can be an hard to solve depending on the constraints and assumptions considered. To simplify this task, in this project, any object with a round shape and roughly 42 cm diameter (the Kobuki base size) is accepted. When multiple robots are found, the nearest is the one to follow.

The information from the depth sensor is used and the points at the height of the robot base are filtered. With these, a top view cut is formed, where the walls become straight lines and the robot base a semi circle. The circle Hough Transform is then applied to identify possible circles. Having the center of the Leader, the movement vector can be calculated as the difference between the current and Leader positions.

Because the depth is placed near the top of the Turtlebot, close and at the ground level points are not captured. To solve this and prevent movement jitter, three distance dependent behaviors were implemented: follow, stay and backtrack. With the thresholds roughly of 1.25m and 1.3m receptively.

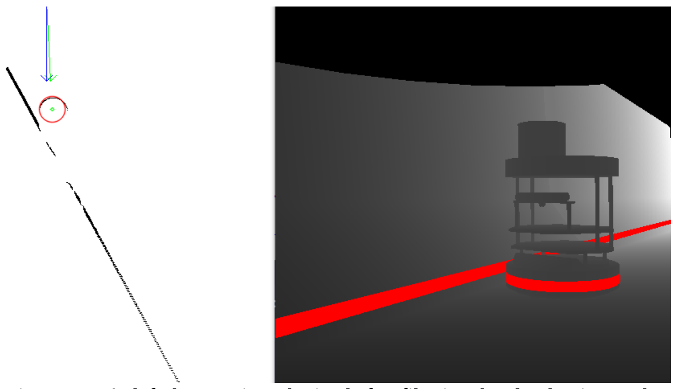
Figure 2. At left the top view obtained after filtering the cloud points at the ground level represented in red on the frontal view at the right. On the top view, the blue arrow is the frontal direction of the robot, the red circle and green dot is the detected robot an the green arrow is the movement vector.

III. Implementation

C. *Tools and libraries*

To develop such system, traditionally, one would have to own multiple robots – in this case Turtlebot2; write the required programs, compile and deploy them to the respective robot and finally run the desired tests on a wide (physical) area. Fortunately ROS encompasses a vast ecosystem that includes tools such as Gazebo – a realistic 3D simulator. Also, because Turtlebot2 is open source, highly connected to the framework development community, multiple packages are available on the ROS repositories, which handle both the access to this robot sensors as well the simulation in Gazebo.

Yet, because of ROS modular architecture and message-based communication mechanism, working with complex messages may be complicated. This is the case of the point cloud messages published by the depth sensor node. Using the Point Cloud Library (PCL), it was possible to quickly iterate through and filter the cloud points, as required by the recognition strategy mentioned above.

Another step of the recognition phase is finding the circle shapes using the circle Hough Transform, which is available out of the box on the Open Source Computer Vision (OpenCV) library.

Debug is an important part of the development process and reaching conclusions analyzing raw numbers on a terminal is a tedious and almost impossible task to perform. OpenCV proved to be a much useful tool to perform this task as it enabled to quickly plot the depth information from the camera and trajectory plans and executions (according to the robot odometry). PCL also provides a visualization tool for the point cloud which ended up to not be used.

Odometry is a not much precise source of information as it is calculated based on the movement of the robot wheels and consecutive slippages introduce an increasing error. To minimize this effect, the *ROS Pose EKF* package was used, that according to its documentation uses “an extended Kalman filter with a 6D model (3D position and 3D orientation) to combine measurements from wheel odometry, IMU sensor and visual odometry”, obtaining a more accurate signal.

Finally, writing code may become more complex than it should when using a plain text editor. After some exploration with the Integrated Development Environment (IDEs) recommended in the ROS homepage, the Qt Creator, with the ROS Qt Creator Plug-in, offered by the ROS industrial consortium, was the environment chosen. It integrates with the catkin package structure and offers all the standard IDE features: text highlight, code completion and compilation.

D. *Development process*

The first step of the development phase was the integration of Turtlebot2 and Gazebo. Because of the multiple nodes required for each robot, the usage of ROS namespaces was required to avoid conflicts between same purpose nodes of different robots.

The required topics to implement the specified capabilities were then identified: <namespace>/cmd_vel_mux/input/teleop used to control the movement velocity of the Turtlebot2, <namespace>/camera/depth/points to obtain the depth points from the Asus Xtion Pro sensor and <namespace>/odom to obtain the robot odometry – latter replaced by the <namespace>/robot_pose_ekf/odom_combined.

After much experimentation with PCL, OpenCV and PointCloud2 message a first version of the following capability was available. Testing and debugging was performed by using the gazebo_teleop package to control a dummy Leader. Then, the Leader was implemented and the two were tested together.

Finally, to evaluate the system under a more realistic scenario than empty space, a Gazebo map mimicking a urban environment was designed.

Work developed at the Robotics course 2016/17, MIEIC/FEUP.

E. *Produced packages*

With this work mainly two ROS packages were produced: **ducklings_gazebo**, which contains the launch files and world maps; and **ducklings_follower** which contain the two C++ programs: listener (Follower) and walker (Leader).

IV. Evaluation

F. *Evaluating walking predetermined trajectories*

In order to evaluate the solution purposed for the problem shown in the figure 2, the turning movement was highly conditioned (1% of the required) and the results are presented in the Figure 4. In this scenario, the robot cannot follow the plan and constantly deviates outpacing the plan. The robot stops, which can be seen by the clustering of position samples. Once the robot steps ahead of the planned position, it will try to move towards it again.

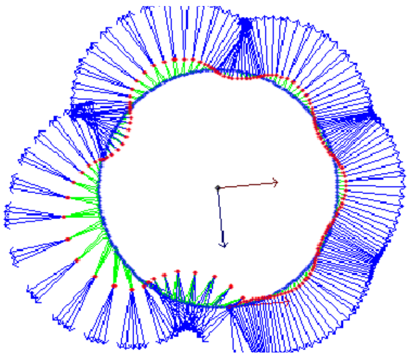

Figure 4. Robot consecutively tries to converge into the planed trajectory.

No collision avoidance was implemented, and this scenario presented in figure 5 illustrates this scenario. First the robot slightly, collides tangentially against the barrier and without much difficulty continues the plan. Yet, when it collides frontally with the cone it gets stuck while in its odometry based view, it keeps moving forward result of the slipping of its wheels.

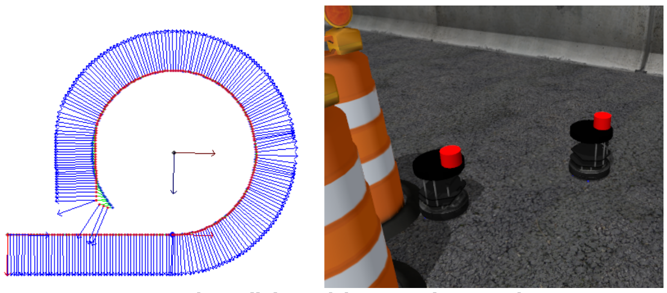

Figure 5. Robot collides with barrier and gets stuck.

As stated previously, traveling in an eight-shape figure is possible by combining two ellipses, as illustrated in figure 6. However, in the same figure, it is possible to observe the small error after the first iteration. This is due to the small error in the robot's orientation at the end of each primitive execution.

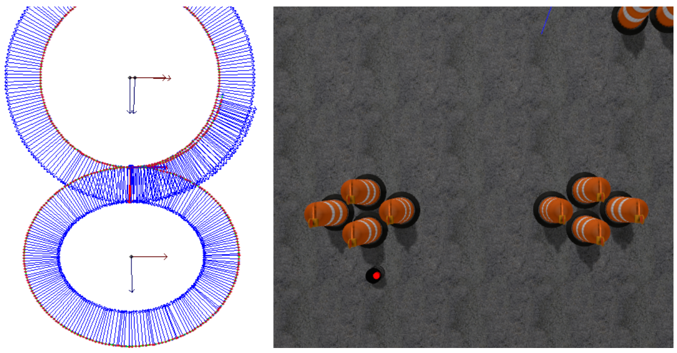

Figure 6. Robot performs a eight shape trajectory.

G. *Evaluating following trajectories*

As it should be expected, because of the fact that followers consider only the (next) Leader position, when performing curves they shorten the trajectory. Also, when multiple following in third or higher position, it is possible that the robot misses the recognition of the nearest leader and attempts to follow a leader ahead. This is visible in the figure 6 where the second follower is orientated towards the first leader instead of the second.

each primitive could be solved by considered the last planned orientation. Following performs as expected.

H. *Future work*

Although it was already experimented in this work, due to time constraints this ended up not being fully implemented and not included in this paper: watching for each other and stopping when one stops. Because the depth sensor only covers the frontal view, it is required that each robot stops its trajectory, turn around it self until finds its subordinate and then returns the planed trajectory. To perform this operation it would be convenient to detect not only the robot position but its orientation, so that one could wait until its subordinate is ready to follow him, before returning the plan. This could be achieved by placing an slightly offset cylinder on the top of the robot and measure the displacement between its center and the base center. These extra marks could be also colored and used to distinguish robots from each other preventing the case observed in Figure 6.

REFERENCES

[1] M. O. K. , Jason “A Gentle Introduction to ROS”, October 2013, ISBN 978-1492143239, Independently published