Turtlebots Parade Report

A project of the 2017 Robotics Cource of the School
of Information Science and Technology (SIST) of Shanghaitech University

https://robotics.shanghaitech.edu.cn/teaching/robotics2017

Zhang Yao
Student ID: 46098322
Email: zhangyao1@shanghaitech.edu.cn

Hu Junyu
Student ID: 31983649
Email: hujy@shanghaitech.edu.cn

Min Jie
Student ID:10109867
Email: minjie@shanghaitech.edu.cn

***Abstract*—This project is mainly about using many turtlebots to generate a parade. A parade is a prescribed rule of following between swarm robots. We want to establish many seperate individual robots which can form a swarm group so that the main task involved is about recognition and following of the turtlebots. In the project, we accomplished the goal that make three turtlebots driving in a sequence.**

I. Introduction

Swarm robotics is an approach to the coordination of multirobot systems which consist of large numbers of mostly simple physical robots. It is not only seems interesting to see a group of robots moving as a team, but also a very important part of robotics field and has many potential applications. For example, we may need many robots to go to a certain place to complete some missons such as searching and rescue, but it is difficult and also annoying if we need control all the robots' trajectories together at one time. In the meanwhile, it is also very challenging to plan for a group of robot because most of our knowledge about robotics is at individual robot level.

Therefore, our plan of the project basically contains the fundamental movement of swarm robots like following or queuing. The basic idea is using a robot as a leader and others are following its trajectory based on some rules. We may start with two robots and expand to three robots case.

We now can make three turtlebots driving in a sequence around a about 3 metres by 3 metres square trajectory relatively smoothly.

II. State of the Art

A. Related Work

In [12], it presents a new method for coordination of a group of mobile agents that can be used for unknown area exploration and monitoring. The idea is inpired by ant colony optimization and some previous research like virtual bird flocking. The main idea is to set the unexplored as a 2-D array and divide it into $m \times n$ square cells. If a certain cell is explored, its value will be set from 0 to one. Also, it maintains two vectors, $\vec{r_l}$ and $\vec{r_p}$, for each robot, where $\vec{r_l}$ represents the position of other robots and $\vec{r_p}$ represents the positon of the nearest unexplored cell. Then they keep updating the two vectors and the next movement taken by a single robot is determined by $\vec{r_l} + \vec{r_p}$. These two vectors are calculated by some formulation that can ensure every two robots keep a 'safe' distance as well as make all the robots distribute evenly in the whole map. They also made some optimization to the algorithm. For each robot, it only cares the $\vec{r_l}$ of the nearest other robot. They also add limitition to each $\vec{r_l} + \vec{r_p}$ so that the robots won't move too agressively in a single step. Besides, they change the value in each square cell($s[i][j]$) from 0-1 value to a probability value. Therefore, robots are mostly attracted to the space surroundings which was explored, but the longest time ago. According to their simulation result based on their algorithm, it works nicely. This may give us some inspiration of how to maintain a safty distance between different robots and make them move orderly during the parade when there are a large amount of robots.

In [5], it provides us with a complete framework presented for landmark recognition and scene understanding with a combination of data-driven and medel-driven approaches. In the data-driven approach, natural scene analysis is performed using a proposed texture model called intensity interactive maps(IIM). The model-driven approach contains a multilevel Markov Random Field based model and an active contour model that integrates color, texture and shape priors.

In [2], it provides a strategy to find optimal path to an object through swarm robots. They use wireless communication among swarm robots and use LED-LDR and ultrasonic sensors for sensoring. The whole system is based on Master Slave concept where master robot guides the slave robots to choose most prominent optimal path to the object.

In [4], it proposes a distributed self-organizing approach where swarm robots are able to self-organize themselves into complex shapes driven by the dynamic of a gene regulatory network based model.The target shape is represented by the non-uniform rational B-spline (NURBS) and embedded into the gene regulation model, analogous to the morphogen gradients in morphogenesis.

In [6], it describes visual functions dedicated to the extraction and recognition of planar quadrangles detected from a single camera. Extraction is based on a relaxation scheme with constraints between image segments, while the characterization we propose allows recognition to be achieved from different viewpoints and viewing conditions. We defined and evaluated several metrics on this representation space: a correlation-based one and another one based on sets of interest points.

In [17], it describes the FTP(The Freeze Tag Problem) and DFP(Dispeerse and Fill Problem).

In [8], it mainly talk about how to increase robustness and efficiency among decentralized swarm robots, improve communications but reduce number of broadcast messages in order to use less power and handle more obstacles scenario.

In [13], it uses probability calculation to optimize detection ability and classify more events with better rubostness.

In [15], it mainly considered the question about how to recognize conspecifics, namely other swarm members, for any one of the swarm member, and it is based on visual recognition. The paper proposed a method by which members of a robot swarm could recognize fellow swarm members and gauge their separation. The method was inspired by the bees, which should be especially attuned to the striped pattern on their abdomen. Bees are able to identify black and yellow, which are the colors of their abdomen, and are also able to distinguish patterns of linear stripes. For this reason, they marked robots with strips of black and white which have the same width. Then they used line-scan vision camera system to detect lines of stripes, This visual recognition system was required to provide the range, bearing, and positive identity of a conspecific robot. Additionally, a moving observer can calculate its distance to a stationary object r given knowledge of its own linear velocity v, the bearing of the object and the object's apparent angular velocity θ'. At last, it would also judge the distance of stripes to ensure the charactistic is from its fellow. Since our goal is to make our swarm robots to form a parade, it is important for them to recognize their fellows. This article introduced an approach to realize it. Although its method just detecting a simple charatistic, where we may want to recognize fellows with as less as possible additional marks, it is still a good guide for our starting.

In [7], where they developed a reliable and fast visual recognition algorithm to detect kin object in a robot swarm, where every robot was equipped with a zebra pattern. The key idea is based on Fast Fourier Transform (FFT), which is sensible for this kind of pattern and has a relatively low complexity.

In [3], it presents a controller design and hardware specifications of robot for SWARM application using Arduino MEGA-2560. Multi Robot Communication is implemented to achieve Leader-Follower approach of SWARM navigation where leader robot guides the slave robots.

In [1], it presents an appearance based method for place recognition. The method is based on a large margin classifier in combination with a rich global image descriptor. The method is robust to variations in illumination and minor scene changes and evaluated across several different cameras, changes in time-of-day and weather conditions.

In [18], it proposes an advanced PSO with the mutation operator, which can solve the local minimum problem to a certain degree. It also presents a novel approach of path planning. First the MAMINK graph is built to describe +e working space of the mobile robot then the Dijkstra algorithm is used to obtain the shortest path from the start point to the goal point in the graph,finally the particle swarm optimization algorithm is adopted to get the optimal path.By adding a mutation operator to the algorithm, it can not only escape the attraction of the local minimum in the later convergence phase, hut also maintain the characteristic of fast speed in the early phase. The results of the simulation demonstrate the effectiveness of the proposed method, which can meet the real-time requests of the mobile robots navigation. Aiming at the shortcoming of the PSO algorithm,that is,easily plunging into the minimum,this paper pots forward an advanced PSO algorithm with the mutation operator. Particle swarm optimization PSO is an evolutionary computation technique developed by Dr.Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling. PSO is easier to implement and there are fewer parameters to be adjusted. After modeling the robots working space, the path-planning problem translates into a shortest path-finding problem in the MAKLINK graph, which can be solved by the mature algorithm in graph theory. But the robot can walk along the edges of obstacles rather than must dong the network path. Therefore,the shortest path of the MAKLINK graph is not the exact optimized path of the whole planning space.

B. Ros Packages

1) kobuki_node: [10]Since we are using kobuki robots for this project, this package provides us with a control loop and ros api for the kobuki driver. We just need to run *roslaunch kobuki_node minimal.launch* to start the robot. During the runtime of the robot, it keeps publishing many topics like the bumper condition or its local position. In the meanwhilie, it also has lots of subscirbed topics with which we can easily determine where to drive the robot. Actually we only use part of these.

First we keep subscribing the *odom* topic for the robot itself and its leader. This topic store the pose and twist messages which represent the position and orientation repectively. It sets the start point of the robot as the origin point whenever we run the kobuki node. Thus we should always restart the kobuki_node to reset the position information. What we need here is actually a 2-D coordinate which we will use for calculating the transformation between different robots later.

Another topic we are using is $command/velocity$ which subscribes $geometry\_msgs/Twist$ message. It has a linear velocity as well as an angular velocity, so we only need to publish the speed we want to this topic and the robot will move in the corresponding speed. We find that the change of speed is really fast , so we add an 'accelerated speed' by hand in case the laptop falls off the robot when a sudden stop occurs.

2) freenect: [9]This package contains launch files for using a Microsoft Kinect using the libfreenect library. We use this for starting the camera (Kinect) on the kobuki robot. We just run $roslaunchfreenect_launchfreenect.launch$ to launch it.

After we start the kinect by using freenect, it mainly recevies three different types of image: depth, ir and rgb. We only need the rgb image here for the recognition of the apriltag (which will be convered later). The good news is we don't really need to subscribe or publish anything when using this package because the $apriltag\_node$ will communicate with the camera automatically as long as we start both of them.

3) tf: [11]tf is a very powerful package that can help us keep track of mutiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time. We need this because we should ensure each robot knows the position of its leader in its coordinate system so it can keep following the leader. Appearantly this is all about transformation between the coordinate frames of different robots.

This package actually provides us with listeners and broadcasters for transform so that we can easily exchange information between different frames. But actually we handle it in a simpler way. We just create transform objects for different objects (including robots and apriltags) given their position and calculate their transform relationship accordingly.

4) Apriltag_ros: [16] [14]AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera. The AprilTag library is implemented in C with no external dependencies. We attach these tags at the back of each robot so other robot can easily identify it.

The $apriltag\_ros$ package provides us with a useful topic called $tag\_detections$ that we can subscribe it as long as it is launched. This topic contains an array of poses including the tag id for each pose. From this we can easily get the orientation of the tag and distance between the camera and the tag in front of it as long as the tag is in sight. Otherwise, the array will only contains null pointer. So we can judge if the camera catches the tag based on this. After we get the position information of the tag with respect to the camera, we actually know where the leader robot is. Thus we only need to calculate the transforms between different robots based on the data provided by this topic. Note that we actually need another two transforms that are from the camera to the robot and from the tag to the robot, but $apriltag\_ros$ doesn't provides us with these. So our solution is measuring it by hand since we assume that they are only translation transforms.

III. System Description

A. System Establishment

For the implenmentation, we choosed the turtlebot 2 of kobuki which is a low-cost, personal robot kit with open-source software. The TurtleBot kit consists of a mobile base, 2D/3D distance sensor, laptop computer or SBC (In this project, we use our laptops as the controllers), and the TurtleBot mounting hardware kit. We used this since it is relatively easy-controlled to implentment our ideas so that we can put more efforts on algorithms such as planning design.

In this project, we use three laptops to handle the three-robot problem: each laptop control on robot. Three robots will walk in a line: one as the leader, the other two will follow one by one.

Furthermore, we also need to put Apriltags on our robots to indicate its identification, as is mentioned in above section, the Apriltags will also give us the relative position between each pair of leader and follower. We put the Apriltags at the back of the robots so that the follower robots can see the leader robots. Figure 1 shows the example of a pair of leader-follower system.

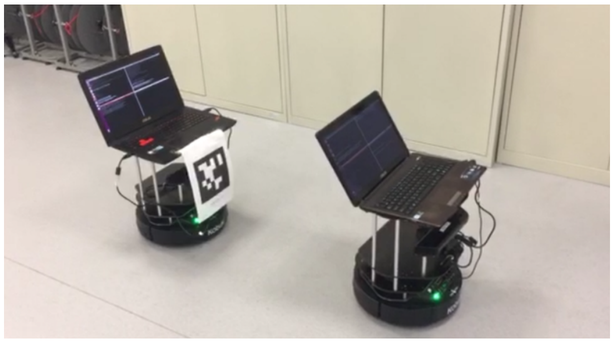

Fig. 1. Here is the overview of the leader-follower system: the follower robot and the leader robot with the Apriltag used to help the follower recognizes it.

B. Algorithm and Code

To make the follower robot follows the leader robot smoothly, our algorithm can be divided to two parts.

The first part is when the follower robot can see the Apriltag of the leader robot. In this certain scene, we do the control of the follower robot just based on the position of the Apriltag in its view: we always try to keep the position of the Apiltag in the middle of its view with a certain distance. Therefore, the linear speed of the follower depends on the distance between Apriltag and itself while the angular speed of the follower depends on the deviation of the Apriltag from the middle. The Algorithm 1 below will show the idea of the above.

Algorithm 1 FollowTheTag

```
1:  if $pose\_tag$ leftTo $center\_pose$ then
2:      $speed\_angular \leftarrow p_a \times pose\_tag$
3:      $speed\_linear \leftarrow p_l \times distance$
4:  else if $pose\_tag$ rightTo $center\_pose$ then
5:      $speed\_angular \leftarrow -p_a \times pose\_tag$
6:      $speed\_linear \leftarrow p_l \times distance$
7:  else
8:      $speed\_angular \leftarrow 0$
9:      $speed\_linear \leftarrow p_l \times distance$
10: end if
```

The other part is when the follower robot can not see the Apriltag of the leader robot. This may happens when there are some obstacles or the leader makes a sudden, large turn. We want to keep the follower follows the leader in such a scene happens. To do this, we need always compute and record the transformation between the global reference of the follower and the global reference of the leader when the follower can still see the position of the leader. In the ideal scene, we only need compute the transforamtion one time since it is a fixed value, however, considering the errors caused when the robots moving, we need always fresh our transformation. When the follower can not see the Apriltag of the leader, we will use the latest transformation we computed.

To compute the transformation and the position the follower should go when it can not see the Apriltag of the leader, we also need transfer the message the position of the leader in its own global reference to the follower, which means we need make communication between the robots. The method we use to make communication between the robots is using a common ROS MASTER, and for each robot, they need a individual namespace to avoid namespace conflicts. We assign the namespace just based on the robot ID we pre-determined.

```
<group ns="$(env MY_ROBOT_ID)">
...
</group>
```

The algorithm 2 below shows that how we compute the transformation in a easy way. Now for the follower, it can trace the trajectory of the leader and keep following the leader by some controls.

Algorithm 2 ComputeTransformation

```
1: $T^{follower}_{follower\_global} \leftarrow pose\_follower$
2:
3: $T^{leader}_{follower} \leftarrow pose\_tag$
4:
5: $T^{leader}_{leader\_global} \leftarrow pose\_leader$
6:
7: $T^{leader}_{follower\_global} \leftarrow T^{leader}_{follower} \times T^{follower}_{follower\_global}$
8:
9: $T^{leader\_global}_{follower\_global} \leftarrow \textsc{Inv}(T^{leader}_{leader\_global}) \times T^{leader}_{follower\_global}$
```

IV. System Evaluation

A. Main Idea

The propose we want to make is to make our swarm robots form the parade and maintain it with a sufficiently long distance and a stable speed. Therefore, the performance testing we want to make is mainly pointing at the distance and the speed, as well as the duration time and it's stability under artificial random destructive tests (Such as suddenly change the moving direction, block tags, etc).

B. Test Indicator

1) Distance between each robots in the parade: This indicator is meaningful since it indicates the stability of the system to some extent. This indicator is tested under the circumstance that the leader of the parade walks a 3 meters by 3 meters square with the speed of 0.1 m/s and 0.2 m/s respectively, and we have two followers one by one. Figure 2 shows the system when we test this indicator.

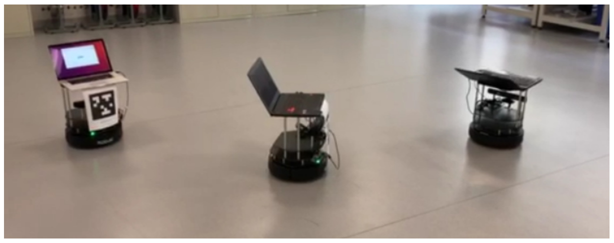

Fig. 2. This figure shows the scene that the leader is running a 3 meters by meters square, and it has two followers one by one.

In the case of 0.1 m/s speed, the initial distance of three robots is 0.4m (from the front of the follower to the back of the leader). We run the system 5 times, each time we let the leader run 5 rounds of the square (about 10 minutes each time). During one times running, the maximum distance between the leader and follower is about one meter (from the front of the follower to the back of the leader).

In the case of 0.2 m/s speed, the initial distance of three robots is 0.6m (from the front of the follower to the back of the leader). We run the system 5 times, each time we let the leader run 10 rounds of the square (still about 10 minutes each time). During one times running, the maximum distance between the leader and follower is about 1.5 meters (from the front of the follower to the back of the leader).

2) The rotation angular of the leader: This indicator is to test the largest rotation angular the leader takes turn when the follower can still follows the leader. This indicator is meaningful since the most tricky part of the following is when the leader has a big or sudden turns. The shape of a turn can be described by the rotation angular and the rotation radius. Since the rotation radius is hard to estimated and it is determined by the linear speed of the robot, we consider the two kinds of linear speeds' case in the experiment. Figure 3 shows the case when the leader is taking a turn.

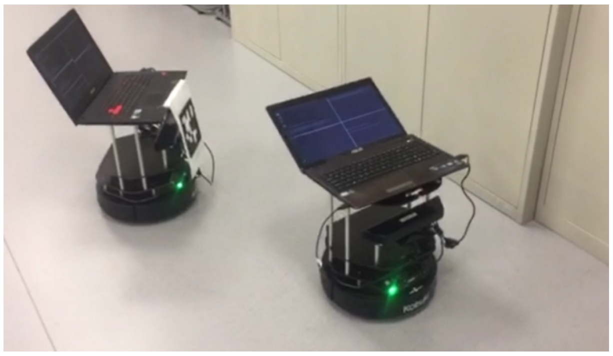

Fig. 3. This figure shows the scene that the leader is taking a turn.

In this test, we fixed our leader's angular speed at 0.3 rad/s. To control the rotation angular, we control how much time does a turn take. We started with 3 seconds as the first trying (that is about 51.6 degrees), and each time we add one more seconds (that is, add about 17.2 degrees each time). Besides, to control the rotation radius, we let the linear speed of the leader be 0.1 m/s and 0.2 m/s respectively.

In the case of 0.1 m/s speed, the largest angle it can follows will between 7 and 8 seconds cases. When the rotation time is 7 seconds (about 120.3 degrees), we tried 5 times and at 2 times it fails to follow while if the rotation time is 8 seconds (about 137.5 degrees), we tried 5 times which all fails.

In the case of 0.2 m/s speed (the rotaion radius will be larger), the largest angle it can follows will between 8 and 9 seconds cases. When the rotation time is 8 seconds (about 137.5 degrees), we tried 4 times and at 3 times it fails to follow while if the rotation time is 9 seconds (about 154.7 degrees), we tried 3 times which all fails.

3) Random walk of the leader: We designed this part since the 3 meters by 3 meters square trajectory maybe seems relatively regular. Therefore, we tried to use some different, more complicated trajectories. We use joy node provided by ROS to control the leader by joystick. We control the the trajectory of the leader as randomly as we can: sometimes fast and sometimes slow, sometimes left turn and sometimes right turn. We record each time how long can the follower can keep following. We tried this 2 times, at the first time the parade kept about 20 minutes and for the second time the parade kept about 17 minutes. In the first time, the second robot lose its leader when the leader took a fast left turn and in the second time, the last robot's distance to its leader (the second robot) became longer and longer after a sequence of continuously turns.

4) Small artificial disturbance: To test the stability of the system further more, we tried to add some small artificial disturbance to the system to how it performs, this part of testing is under the circumstance that the leader of the parade walks a 3 meters by 3 meters square with the speed of 0.2 m/s, and we have two followers one by one. The two following small disturbances we tested have better perfomance than others:

1) Suddenly change the moving direction of a random robot (including the leader and the followers)
   As shown in the Figure 4, we changed the direction of the robot to see whether it can still follows. The system can fix the disturbance if the forced change is not too large.

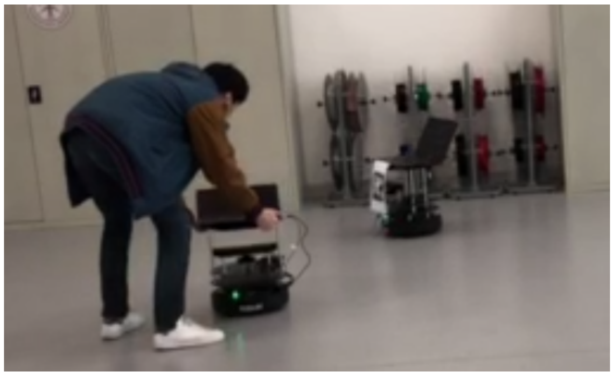

Fig. 4. This figure shows the scene we add a samll disturbance to the directions of the robot.

2) Block tags of a random robot (for followers)
   As shown in the Figure 5, we go through between the robots to and for again and again. The system can fix the disturbance if the disturbance not too frequently.

In the above test, our system can endure some small disturbance. Although it cannot handle too much disturbance, the performance is relatively accepted.

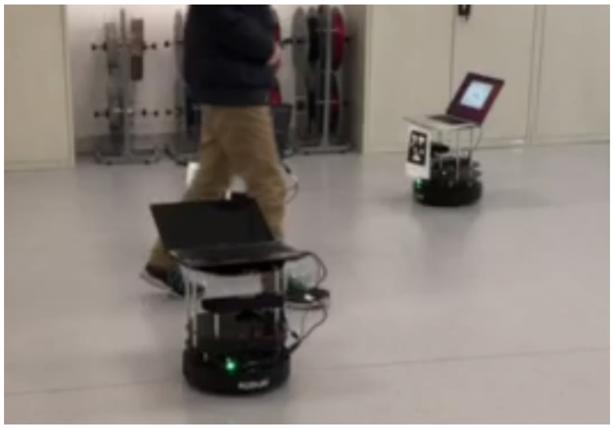

Fig. 5. This figure shows the scene that we go through between the robots parade.

C. *Elementary Result*

For result, our parade robots could form the troop in short time, and it's quiet stable for reasonable range of speeds. The system is quiet robust under some small artificial random disturbance tests, although it may not be ensurable to some more destructive circumstances.

V. Conclusions

A. *Summary and Conclusion*

Our Project is mainly about using many turtlebots to generate a parade and maintain it through a map. The parade is basically formed by a group of robots that one followed by another one by one. Also, the routes we set include both clockwise circle and counterclockwise circle. The measurements we use to test our stability and performance is mainly time of duration, which means how long the robots can maintain the parade nicely at most. We only use three robots for testing. Since the path is not very complicated, a parade with three robots should be convinced enough.

Our recognition method uses the Apriltags, and the following algorithm is divided into two parts. Different method is called for control depends on whether the follower can see the Apriltag of the leader. Additionally, we can also get the id of the Aprialtags, so we can ensure that each robot follows its own leader correctly and won't be misled by the other ones.

We test the robustness of our algorithm under the speed of 0.1m/s and 0.2m/s respectively. The algorithm can also handle some unexpected condition like a sudden change of direction by human but it may fail when being disturbed too hard or too frequently.

B. *Future Work*

In the following part, the things we need to do are:

1) SBCs: We may establish the environment of ROS on the Tinker Board to control more robots without laptops.

2) Improve Algorithms: Although our algorithm works out well when the disturbing factors are limited, it need to be improved when it comes to a bigger disturbing. What's more, we give up only tracking the leader's trajectory because of a significant accumulation of error. If we can find out a better way to handle this error, the algorithm may have a better result and work on a more complicated route of the parade.

3) More Robots: We may add more robots into our parade. All we need are more laptops with a copy of codes. We may also add different robots in rather than only turtlebots but the parameters for the algorithm need to be modified.

References

[1] P. Jensfelt A. Pronobis, B. Caputo and H.I. Christensen. A discriminative approach to robust visual place recognition, 2006.

[2] Dr. C.S. Satsangi Ankita Saxena and Abhinav Saxena. Collective collaboration for optimal path formation and goal hunting through swarm robot, 2014.

[3] F. S. Kazi Dhiraj Arun Patil, Manish Y. Upadhye and N. M. Singh. Multi robot communication and target tracking system with controller design and implementation of swarm robot using arduino, 2015.

[4] Yan Meng Hongliang Guo and Yaochu Jin. Swarm robot pattern formation using a morphogenetic multi-cellular based self-organizing algorithm, 2011.

[5] Fritz J. and Dolores H. Russ. Generalized landmark recognition in robot navigation, 2004.

[6] F. Lerasle J.B. Hayet and M. Devy. Visual landmarks detection and recognition for mobile robot navigation, 2003.

[7] T. Kovacs K. Bolla and G. Fazekas. A fast image processing based kin recognition method in a robot swarm, 2014.

[8] Anamika Lal. Efficient robot swarm movement and shape recognition algorithms in environments with obstacles, 2009.

[9] Rospackage List. freenect_launch, http://wiki.ros.org/freenect_launch.

[10] Rospackage List. kobuki_node, http://wiki.ros.org/kobuki_node.

[11] Rospackage List. tf, http://wiki.ros.org/tf.

[12] M. Masar. A biologically inspired swarm robot coordination algorithm for exploration and surveillance, 2013.

[13] Matthew R. Proffitt. Optimization of swarm robotic constellation cmmunication for object detection and event recognition, 2011.

[14] RIVeR-Lam. https://github.com/river-lab/apriltags_ros, 2017.

[15] R. Andrew Russell. Visual recognition of conspecifics by swarm robots, 2012.

[16] AprilTags Visual Fiducial System. https://april.eecs.umich.edu/software/apriltag.html, 2016.

[17] Marcelo Oscar Sztainberg. Algorithms for swarm robotics, 2003.

[18] Ning Li Yuanqing Qin, Debao Sun and Yigang Cen. Path planning for mobile robot using the particle swarm optimization with mutation operator, 2004.