Improving Object Detection for the Turtlebot People Follower Using AprilTags

1st Patrick Herke
Mechanical Engineering Student
Lousiana State University
Baton Rouge, US
pherke1@lsu.edu

2nd Evan Nguyen
Biological Engineering Student
Louisiana State University
Bation Rouge, LA
enguy21@lsu.edu

3rd Julius Pallotta
Mechanical Engineering Student
Louisiana State University
Baton Rouge, Louisiana
jpallo1@lsu.edu

***Abstract*—This paper reports on the improvement of the follower code for the TurtleBot. The improvements made relate to distractibility, range of detection, ability to find new targets, and addition of audio indicators for whether it is actively following. The new code uses fiducial markers in the form AprilTags (similar to simplified QR codes) to identify and follow an object using a TurtleBot. The original follower code used the depth camera to find the centroid of a bounded region of the image and was thus very easily distracted. By replacing the current object identification system with AprilTags, the robot can differentiate between the target object and interfering objects with a near 100% success rate as opposed to the random chance of the previous code. The new code can pick up on a new target at roughly double the range. The TurtleBot now turns in place with an intermittent nature to it if it has not seen a target for 5 seconds to improve its chances of finding a new target. In addition, audio indicators for stopping and starting have been added.**

***Index Terms*—follower robot, fiducial markers, kinect sensor**

I. Introduction

Tracking software has many applications in robotics. Some of these potential applications include mobile storage robots [1], guides in a nursing home [2], or exploring caves alongside people [3]. Following robots that have facial recognition technology can also be used in places like hospitals or airports to recognize a person and escort them around or bring them their luggage. These robots could also be used as an autonomous shopping cart that follows you everywhere and puts itself back in the proper places when not in use. There are also flying drone type robots that have object detection and subject recognition [4] to assist in picture and video capture.

This project aims to improve the original people follower's code for the TurtleBot. The TurtleBot consists of a Kobuki base, an ASUS Xtion Pro Live camera, and a HP laptop that runs Ubuntu and ROS melodic. The ASUS Xtion Pro Live camera has an RGB camera and a depth camera. When evaluating the performance of the original people follower code areas noted for improvement included distractibility, range of detection, ability to deal with abrupt changes in direction, speed of the robot, ability to find new targets, and the addition of audio indicators for whether it is actively following. It was decided that the biggest issue was the distractibility of the old follower code. Thus the primary goal of this project is to modify the follower code to improve the TurtleBot's ability to identify a person or object and successfully follow that person amongst various distractions. For secondary goals the following issues were selected: the range of detection, ability to find new targets, and the addition of audio indicators for whether it is actively following.

The next section will discuss the approach to solving the issues identified in the introduction. This includes further explanation of AprilTags and how they are used in our code. The logic behind the velocity control, sound play, timers, and search behavior will also be discussed. When discussing coordinates relative to the camera, from the perspective of the camera, the x-axis is positive to the left, the y-axis is positive upward, and the z-axis is positive forwards.

II. Approach

This project began with the TurtleBot people follower code that can be found on GitHub. By using that preexisting code instead of creating one from scratch, time and computational resources were conserved. The primary downside to using preexisting code is the time it takes to understand the methodology of the original authors. After testing the code with the robot, we analyzed the code to understand its logic. This took a considerable amount of time given that the code was only sparsely commented.

The previous code simply found the centroid of all the points within a box centered around the center of the depth image. Figure 1 shows a representation of the depth image as seen in rviz. It made no distinction between different objects. This caused it to be prone to changing targets if anything except the person it was following appeared within that box in the image. We first considered if there was any easy way we could modify the way it found the centroid of the points in order to make it less prone to swapping targets. It was concluded that this would require some way to differentiate the objects and store that differentiating information. Once that was accomplished it would be possible to find the centroid of the points associated with that object and not any others.

Three different methods of differentiating people were considered: object detection via skeletal diagrams, color detection, and fiducial markers. Upon considering how each might be implemented it was decide that the most reliable method would be to tag a person using fiducial markers. Object detection

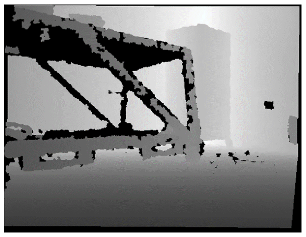
Fig. 1. The depth image from the ASUS Xtion Pro Live camera seen in rviz.

solves the issue entirely but it takes too much time to build a system that accurately identifies a human or object that the robot will then follow. This is because the method uses machine learning to differentiate between different objects and stores that in a library where then it is specified what object to follow. Color detection runs into the same problem as the original follower since it does not help differentiate between two people wearing the same color. Fiducial markers avoid these pitfalls because the user can specify their appearance to high degree. As such they don't need complicated machine learning to detect and are unlikely to occur naturally in the environment.

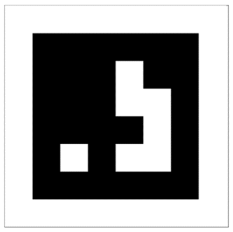
Fig. 2. A simpler AprilTag from the 16h05 family.

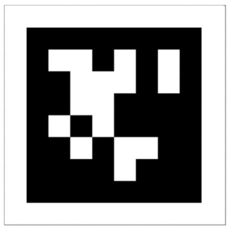
Fig. 3. A more complex AprilTag from the 36h11 family.

AprilTags are a particular type of fiducial marker that have been created to specifically to provide accurate data on their position and pose relative to the frame of the camera. They are relatively robust to different lighting conditions and view angles due their limited complexity. However, they contain enough complexity that natural features of the environment are unlikely to be mistaken fro AprilTags. The nature of AprilTags eliminates the need for a depth camera to determine distance from an object. There are multiple families of AprilTags that can be used. The variations in complexity affect the range at which they can be detected and the false positives rate with more complex tags have lower ranges and lower false positive rates.

The code for finding the centroid of the point cloud was replaced with code for identifying the center and orientation of the AprilTag. The continuous detection program from the ROS package apriltag_ros [5] is used to continuously search the RGB camera feed of the TurtleBot for AprilTags of a specific family and id. This program is able to find the orientation of the AprilTag based on how much of the image it occupies and to appearance of the squares which make up the image. It compares this information to the size that the configuration file tells it the AprilTag should be. The follower code accesses this information by subscribing to the tag_detections topic which the continuous detection program publishes to. This process handles multiple tags in frame by averaging the locations of all the AprilTags detected. Averaging the AprilTag locations was chosen because the locations are all relatively close together, as can be seen in Fig. 4. Different behavior, such as prioritizing certain tag ids might be implemented if different configurations of tags were used.

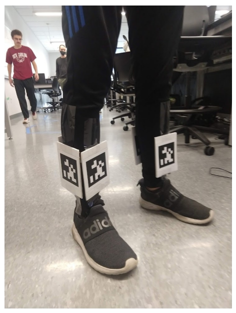
Fig. 4. Setup of tags on the person.

The goal distance for following was set to be 0.5 meters. Based on how far the tag is from that goal distance the robot adjusts its forward velocity via a the proportional control in (1). The angular velocity of the TurtleBot is determined via a the proportional control in (2). The follower code interacts with the velocity by publishing velocity commands to a topic. If no tags are found, the TurtleBot stops moving until it sees another tag. Unlike the original TurtleBot people follower code, when someone passes in between the robot and the target human the robot will not switch targets. This is because the TurtleBot is looking for a particular AprilTag rather than simply finding the centroid of the point in bounded box.

$$z_{vel} = (z - z_{goal}) \cdot z_{scale} \tag{1}$$

$$x_{ang} = -x \cdot x_{scale} \tag{2}$$

If the TurtleBot doesn't see an AprilTag it will indicate it has lost the target by emitting a sound. It also indicates that it has found a new target to follow by emitting a different sound. In each case, the sound only activates if the new state has occurred for at least 0.5 seconds. This is to eliminate the scenario where the robot only briefly either loses track of or finds a tag. In such a scenario without the delay the TurtleBot would end up making a stopping or moving sound immediately followed by the other audio indicator. Eliminating that situation improves the reliability of the audio indicators in giving the people around the robot useful information about the current state of the follower code without introducing a lot of extraneous information.

The first attempt on audio cues was to use ROS packages already available, like sound_play, but with the inability of the node to produce sound to the laptop the idea was abandoned. The next thing that was attempted was to instead open a sound file using code in the follower.cpp file. This idea was also scrapped after some time because the included libraries needed to run those commands are not supported in Linux. In the end the built-in beeping sounds that the Kobuki base comes with were used. This was accomplished by publishing messages to the /mobile_base/commands/sound topic specifying which beep pattern to play. There were a total of 7 patterns available and we used $value = 6$ for the stopping of the robot and $value = 5$ for the starting of it. These tones were only played when the robot found or lost the tag.

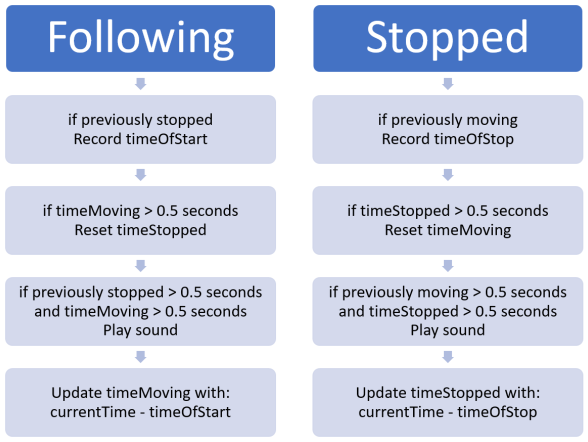

Fig. 5. Shows the logic flow of the timer functions as they relate to keeping a record of the time stopped and the time moving.

To improve the TurtleBot's ability to locate new targets, if it didn't see an AprilTag for 5 or more seconds it starts to rotate counterclockwise intermittently until it spots a new target again to follow. The intermittent motion helps give the TurtleBot enough time to pick up on an AprilTag that has entered its field of view and then stop turning before it loses the tag. The logic in the code that determines the intermittent motion is that the TurtleBot only rotates if the decimal portion of the time it has been stopped is less than 0.5 seconds. Thus the ratio could easily be adjusted for better detection by changing the 0.5 seconds value to adjust the ratio of stopped to moving, dividing the decimal portion by some constant to make the period smaller than a second, or adjusting the turning speed.

The timing was done using the second and nanosecond time stamp of the messages from the tag_detections topic. Using a series of if statements the time was recorded in a series of different variables (timeStart, timeStop, and currentTime) and used to update other variables (timeMoving and timeStopped). The times recorded were then used to determine if sound should be played and if the robot should begin searching for new targets. The following section will discuss how the success of the new follower code was evaluated.

III. Experiments

Three experiments were conducted to evaluate the success of the improvements to the original people follower code. For each experiment, the setup of the person with six April tags of id=0 and tag family=36h11, was used. The first experiment found the maximum distance that the TurtleBot could reliably detect and begin to follow a new target. The second experiment tested the ability of the TurtleBot to pick up on a new target while it was rotating to find a new target. The third experiment tested the distractibility of the new versus the old follower code.

The first experiment found the maximum distance that the TurtleBot could reliably detect and begin to follow a new target. Both the new and the old follower code were tested with three trials run for each. To test the maximum distance, as can be seen in Fig. 6, a person wearing an AprilTag (id #0 from the 36h11 family) stood out of range directly in front of the TurtleBot. After launching the follower code, the person slowly walked forward in increments of roughly 0.25 meters the robot began to detect a target. For the new code, that means it could detect an AprilTag. And for the old code, that means it detected enough points to find the centroid.

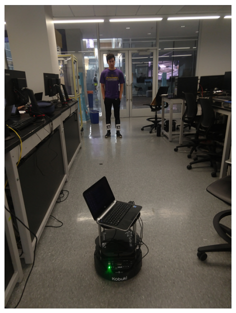

Fig. 6. Demonstration of the setup for finding the maximum distance at which the TurtleBot can detect a new target.

Table I shows that the maximum distance at which the new code could reliably pick up on a target was roughly double that of the old code. The increase in range allowed for the TurtleBot to pick up one a target from a greater distance. This reduces how often the TurtleBot loses track of a target due said target getting too far away from the TurtleBot. It also makes it easier for the TurtleBot to find the target after it has lost track of it because its search space is larger. Some testing was also done with the AprilTag family 16h05. This tag family increased the range to about 4 meters (nearly four times the original amount). However, it was not used in our final version of the project because the software was picking up a lot of false positives when that tag family was used. Both the increased range and false positive rate was due to the 16h05 family being much simpler than the 36h11 family of tags.

TABLE I
MAXIMUM DISTANCE FOR DETECTION OF NEW TARGETS

	Distance (m)	
	New	***Old***
Start	2.67	1.17
Consistent	2.33	1.17

The second experiment tested the ability of the TurtleBot to pick up on a new target while it was rotating to find a new target. The goal of the rotating mode was to look for new targets if the TurtleBot hadn't seen any tags for at least 5 seconds. The initial setup for this experiment is similar to that seen in Fig. 6, except that in this the robot was rotating rather than stationary. The success rate for a stationary person and a moving person at three distances (1 meter, 2 meters, and 3 meters) was tested. For the moving test, the person walked in a circle of a radius of each of the aforementioned distances opposite the direction of rotation of the TurtleBot. This motion is illustrated in Fig. 7. For each combination of movement type and distance, three trials were conducted.

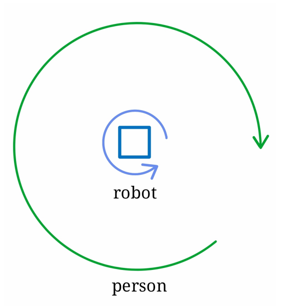

Fig. 7. Movement of the person relative to rotation of the robot.

Table II shows that the success rate for the stationary person was 100% at both 1 meter and 2 meters. However, the success rate declined for the 3 meter distance. This lines up with the results of experiment 1 which suggested a maximum reliable range of detection of between $2.33 - 2.67$ meters. The moving target was detected most frequently at 2 meters, less frequently at 1 meter, and not at all at 3 meters. With moving targets, the detection is unreliable for closer distances and for distances closer to the limit of the range. This is likely a result of the movement decreasing the time during which the tag is in the field of view of the TurtleBot. The improvement in performance at 2 meters versus 1 meters would then be an effect of the TurtleBot's wider field of view at that distance. A potential avenue of improvement would be adjusting the rotational speed and stopping patterns of the rotation.

Lastly, the third experiment tested the distractibility of the new versus the old follower code. Two scenarios were tested. In both scenarios the TurtleBot was actively following

TABLE II
DETECTION WHILE ROTATING

	Success Rate	
Distance (m)	***Stationary (linear)***	***Moving (radius)***
1.0	100%	33%
2.0	100%	67%
3.0	67%	0%

a person. For the first scenario, a second person walked in range of the TurtleBot with a tag from the same family as the target attached to their leg. In the second scenario, an object with a tag from the same family as the target was placed in front of the TurtleBot. Figure 8 shows an example of the second scenario. However, in the actual experiment a different person than the target held the object. As with the first experiment, three trials were run for the new and the old follower code for each scenario. For each trial, the data recorded was whether the robot followed the false target and whether the robot recaptured the original target after its view was obscured.

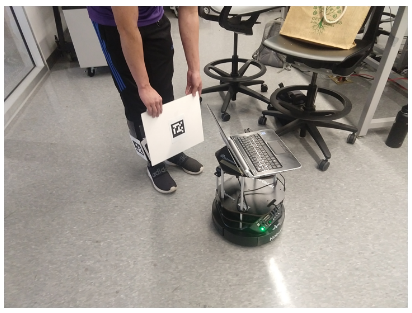

Fig. 8. Attempting to distract the TurtleBot with an object with a wrong tag.

As can be seen in Table III, for the new code, the TurtleBot was not distracted at all and recaptured the target AprilTag every time for each scenario. For the old code, in the first scenario the TurtleBot was distracted on most of the trials and never recaptured the original code. And in the second scenario, the TurtleBot was always distracted and only recaptured once. This indicates that whether the old code continued to follow the same or recaptured the target after removing an obstruction was largely a matter of chance. On the other hand, the use of AprilTags allowed the TurtleBot to avoid diverting to a false target because it could distinguish targets it should follow and ones that it shouldn't.

TABLE III
DISTRACTABILITY

		Occurrence Rate	
		New	***Old***
Person with Incorrect Tag	***Distracted***	0%	67%
	Recaptured	100%	0%
Object with Incorrect Tag	***Distracted***	0%	100%
	Recaptured	100%	33%

IV. Conclusion

The replacement of finding the centroid of a bounded region of a depth point cloud with locating AprilTags greatly improved the TurtleBot's ability to differentiate targets. It went from only being able to stay on target by chance to never being distracted by a non-target object or person. This was true even when the other tags from the same family passed into its field of view. The maximum range at which the TurtleBot could pick up on a new target was roughly doubled. The search behavior that was added to the TurtleBot can pick up on stationary targets that are in range of its maximum range nearly 100% of the time. And it works roughly half the time with moving targets. In addition, the new starting and stopping sound provide people around it audio indicators of its current state where there were none before. These additions significantly improved performance, but there are still yet more changes that could be made.

The AprilTag used in the testing is an 8 centimeter tag from the 36h11 family. Using a tag from the 16h05 family in conjunction with the tag from 36h11 family could further improve the range by another $1-2$ meters. The detection of the 16h05 tags would only be acted on if the robot saw no 36h11 tags and the tag was more than a certain distance away. This would reduce the impact of false positive because the robot would only be considering the 16h05 tag to find a new target. At the same time this use would allow for increased range over just using the 36h11 tag family.

An additional point of future improvement is to tune the rotation during the searching phase. Currently, the TurtleBot completes a full rotation in 22 turns with 1 second intervals and it is rotating for 0.5 seconds out of each second. The ratio of stopped time to moving time, the combined time of one start-stop cycle, or the turning speed could all be adjusted.

Another area that could also be improved upon was the obstacle detection. Currently, if it sees a tag it simply tries to follow regardless of what is in it's path. If a combination of the point cloud and the tag system was used the robot could stop before hitting something. Alternatively, the Kobuki's bumper could be used to identify that there is an obstacle in the way and to stop after running into it.

References

[1] Piaggio fast forward. gita. (n.d.). Retrieved December 4, 2021, from https://mygita.com/.

[2] Fadelli, I. (2020, November 18). A robot that can track specific people and follow them around. Tech Xplore - Technology and Engineering news. Retrieved December 4, 2021, from https://techxplore.com/news/2020-11-robot-track-specific-people.html.

[3] Chakravorty, A. (2021, November 9). Underground Robots: How robotics is changing the mining industry. Eos. Retrieved December 4, 2021, from https://eos.org/features/underground-robots-how-robotics-is-changing-the-mining-industry.

[4] French, S. (2021, November 8). DJI Mavic 3: The closeest we've gotten to a crash-proof consumer drone. The Drone Girl. Retrieved December 4,2021 from https://www.thedronegirl.com/2021/11/04/dji-mavic-3/

[5] Wiki. ros.org. (n.d.). Retrieved December 4, 2021, from http://wiki.ros.org/apriltag_ros.

V. Appendix: Work Breakdown

AprilTag Detection: Patrick and Julius were the ones to work on this section. They both determined what tag families to use, the sizing of the tags, and the placement of the tags. Evan found the package.

Sound: Evan mostly researched this going through all the possibilities and testing them.

Code Implementation: Patrick predominately implemented the AprilTag detection and sound code. Evan assisted a little with the sound code. Patrick was entirely responsible for the determining the logic for the timing aspects of the code.

Presentation: All teammates contributed to the creation of the final presentation.

Report: All teammates contributed to the writing of the final report.

Testing: Julius and Patrick came up with the important criteria to determine the reliability of the TurtleBot and all three team members conducted the tests based on said criterion.

Percentage Breakdown:

- Patrick = 45%
- Evan = 35%
- Julius = 20%