Autonomous Turtlebot Parking with Human Gesture Control

Jeff Hui, Zohair Naqvi, Ryan Peterson

In this paper we design a parallel parking algorithm for the Turtlebot based on detection of ArUco markers in empty spots. Additionally, gesture control is incorporated in the project through the use of a LEAP motion controller device. Gesture recognition, which works off the back of a trained neural network, allows us to switch between different modes of Turtlebot operation: autonomous & manual. This is supposed to simulate actual autonomous parking features available on some commercial vehicles, where an override switch is provided for the driver to take over, if necessary.

I. Introduction

Self-parking features have been available in commercial vehicles for some time now. As early as 2003, Toyota introduced the Intelligent Parking Assist on their Prius hybrid vehicles, which let users automatically parallel park. Since then, with the advent of autonomous vehicles, the need for assisted parking systems has increased exponentially. Currently many companies are working toward fully autonomous vehicles and have begun real world testing with a human in the front seat who has the ability to intervene when necessary.

Whether it be cars or robots working alongside humans in a shipping warehouse, we believe human interaction plays an important role in creating a harmonious and beneficial relationship between users and machines. For instance, if a driver is at the supermarket with their car in autonomous mode the vehicle may choose the first safe/legal parking spot it sees which might be needlessly far from the entrance. Allowing a human to intervene and force the vehicle to find a spot closer to the entrance leads to a better user experience while also alleviating some of the up-front design requirements for an autonomous vehicle since all scenarios cannot be accounted for.

In this project a Turtlebot is used as the self-parking ground vehicle and the user interaction is controlled via a LEAP motion controller that tracks the users hand position and orientation. Certain hand gestures influence the decisions the robot makes, and there is even an option to control the Turtlebot manually with hand position and orientation only. The specifics of which will be outlined in sections to come.

Actual commercial vehicles, however, have an Akerman steering system so the parallel parking controller actuates the steering angle and linear velocity to perform the desired parallel parking maneuver. The Turtlebot, on the other hand, is a differential drive robot which means that performing a parallel parking maneuver is not really required to park in an empty spot. For the purpose of simulating an actual vehicle however, we manually defined a trajectory that resembles a parallel parking maneuver on an actual car.

The rest of this paper is organized as follows: Section II describes the experimental setup and hardware interfacing of the project. Section III describes in detail the Turtlebot's motions and parking algorithms. Section IV describes the testing of different scenarios that the robot's could undertake based on the coded algorithm. Section V discusses the results and includes a link to a video of the successful robot operation. Finally, conclusions and future work are included in Sections VI and VII.

II. Experimental Setup

The project infrastructure consisted of:

- 1 Turtlebot
- 2 Astra cameras
- 1 LEAP Motion sensor
- 1 Turtlebot on-board laptop
- Boxes as obstacles
- ArUco markers

A. Computer vision

Computer vision was an integral part of this project. Two Astra cameras and computer vision algorithms were used to detect the parking spots.

First, the cameras were calibrated using ROS's monocular camera calibration package [1]. Camera

calibration is the process of estimating a camera's intrinsic parameters and is essential for 3D computer vision.

"ArUco" markers were used for the cameras to locate a parking spot. ArUco markers are a type of fiducial marker that can be used to provide a fixed point of reference in the real world. They were generated using OpenCV's ArUco library [2].

The camera calibration process was completed using a chessboard, as shown in Figure 1. An example of an ArUco marker is shown in Figure 2.

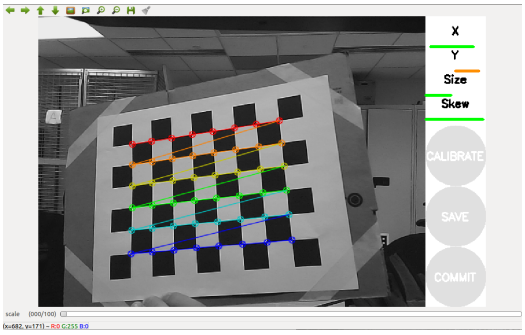

Fig. 1: Camera calibration process

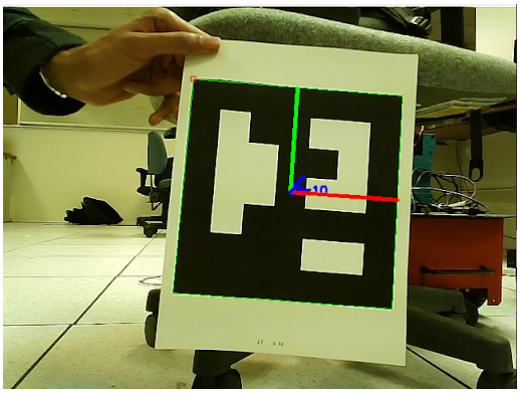

Fig. 2: ArUco marker

B. Physical setup

The experiments were performed in a controlled lab environment. The Turtlebot and both of the Astra cameras on the Turtlebot were connected to a laptop on-board the Turtlebot. This laptop ran both of the cameras and was the means of communication for the Turtlebot to give it velocity commands. The LEAP motion controller was run on a remote desktop with the user. This allowed the user to be stationary and give commands to the Turtlebot via the LEAP motion more effectively. This setup is made possible by connecting to the Turtlebot's laptop via ssh protocol over the network. The setup is shown in Figures 3-5.

For the parking spots, ArUco markers were adhered to the wall between boxes that acted as obstacles or filled spaces, as shown in Figure 3. The Turtlebot was lined up on one end of the room directly facing an ArUco marker on a wall across the room. That ArUco marker was used to indicate when the Turtlebot has reached the end of the row in the parking lot.

Fig. 3: Environment setup: Note there are two spaces between cardboard boxes indicated with ArUco markers, and the ArUco marker on the far wall indicates the end of the row. Shown is the starting position of the Turtlebot.

C. LEAP Motion setup

The LEAP motion controller uses a set of two cameras and three infrared LEDs to capture and reproduce a 3-D reconstruction of the users hand. Information from this reconstruction such as the palm position and hand orientation are published. For this project we used this raw data for manual control of the robot, for instance using the hand pitch angle to control linear velocity of the robot. We also used this data to create recognizable gestures. This was done using the `neuro_gesture_leap` package from GitHub [3]. In order to create new gestures the user simply creates datasets of the published LEAP motion data for a particular gesture over a specified time interval, then those data sets are used as the training data for a Feed Forward Neural Network. This package uses PyBrain [4], which is

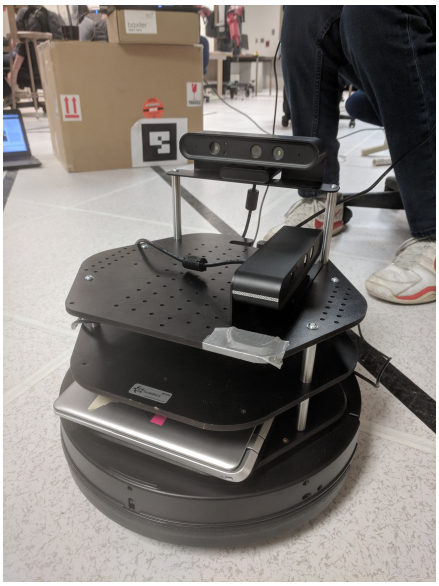

Fig. 4: Turtlebot configuration: contains two Astra cameras and the on-board laptop.

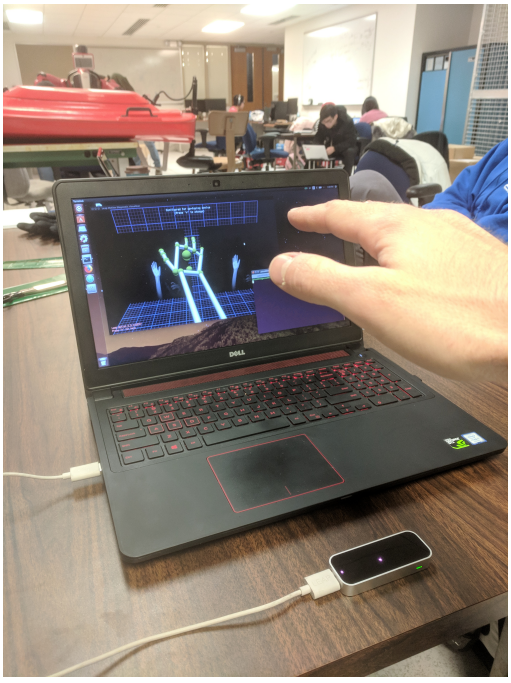

Fig. 5: Remote Laptop configuration: user station where LEAP motion controller runs.

a python based Machine Learning Library, in order to build and train the Neural Network. Once the training is completed the user has a set of robustly recognized gestures that can be published while the LEAP motion is recording user hand data. This is simply published as a String of the gesture name such as 'Swipe' or 'Circle'.

To create our gestures we used 30 training data sets for each gesture recorded on a 2.0 second interval. At a high level this means we will be publishing a gesture every 2.0 seconds that can be used by the Turtlebot navigation framework.

We chose to create four different gestures for the control of our Turtlebot. From our tests, For every gesture the palm was kept down to give more accurate results. We found that the yaw and roll positions of our hands were not recognized as well as pitch. We used a 'Swipe' gesture where the user moves their hand left and right to skip a parking spot, a 'Gandalf' gesture where the user moves their hand up and down to enter manual mode, a 'Hand out' gesture where the user simply takes their hand out of the field of view of the LEAP to enter autonomous mode, and a 'Do nothing' gesture where the users hand is stationary which acts as the default case when the user does not want to perform a gesture.

III. Details of Algorithm

In order for an autonomous vehicle to move through an environment in a safe manner some form of external input is required to observe that environment. For this project we relied on depth and image cameras to perceive the environment. We opted for these since they are readily available at a low cost. Of course using the information obtained from these sensors the Turtlebot needs the ability to act upon that sensor data which in our case is done by controlling the linear and angular velocity of the Turtlebot.

To detect a parking spot and perform a parking maneuver autonomously while also considering human input, we roughly split the Turtlebot operation into four stages.

- Looking for a parking spot.
- Detect an empty parking spot.
- Execute a parking maneuver.
- Move in manual mode.

A. Looking for a parking spot

While the Turtlebot was moving along in either autonomous or manual mode the side camera is constantly looking for an ArUco marker that corresponds to an open parking spot. Meanwhile the front-facing camera detects the distance to an ArUco marker across the room. If the distance detected to this ArUco marker was less than some threshold, in our case 0.7 meters, then this indicates the Turtlebot has reached the end of the ”parking lot.” The Turtlebot then switches into manual mode.

When the Turtlebot is in autonomous mode this simply means the vehicle is moving forward at a constant velocity, and human input for specific velocity control is ignored.

While looking for a spot if the user wishes to enter manual mode they can perform a predefined gesture to allow the user to take over.

B. Detect an empty parking spot

In order to detect a parking spot the side camera has to recognize a specific ArUco marker. This was done by using the ArUco library in OpenCV [2].

Since our parking trajectory was predefined we placed a threshold on when the side camera can detect the ArUco marker, meaning it has to be within a few degrees of being perpendicular to the robot to register. This ensures that the parking trajectory is not performed too early or too late. Once a parking spot has been detected the user is notified. If the user wishes to skip the parking spot they can perform a gesture command using the LEAP motion controller that indicates they would like to skip this parking spot. We chose a swipe gesture for our ’skip spot’ command.

C. Execute a parking maneuver

Once a parking spot has been detected a predefined trajectory is executed automatically to park the Turtlebot in a manner that looks similar to a car performing a parallel parking maneuver.

The user can voluntarily call off the parking maneuver by either performing the ’skip spot’ gesture or performing the ’manual mode’ gesture which allows the user to take over.

D. Move in manual mode

At any time the user can perform the ’manual mode’ gesture to disable autonomous mode and control the Turtlebot manually via hand motion input recorded by the LEAP motion controller. The LEAP publishes topics that can be subscribed to and then converted into velocity commands. We found that hand pitch angle works well for controlling the linear velocity of the Turtlebot. To control the angular velocity we use the x-position of the hand in order to make the Turtlebot turn right or left. To exit the manual mode and enter back into autonomous mode simply take your hand out of the field of view of the LEAP motion controller. This is recognized as an ’Enter autonomous mode’ gesture.

E. Detailed Procedure

For more details on the procedure visit our GitHub page which explains how to install the necessary software and the steps for creating and integrating gesture control for the Turtlebot. `https://github.com/pete9936/turtlebot_parking_leap`

IV. Algorithm Testing

In our tests the Turtlebot started in autonomous mode, so once the test is executed the Turtlebot will begin moving forward at a constant linear velocity. To verify that our self-parking Turtlebot was capable of performing this task while also considering human input as described in the Procedure section we used three test cases.

1) Park in the first spot detected without human intervention.
2) Skip the first spot detected and park in the second spot detected.
3) Skip all spots and go until the end of the course (simulated ”parking lot”). Then, take full manual control of the robot.

V. Results

Through our experiments described in the previous section we fine-tuned many aspects of our navigation protocol. Some of which included the defined trajectory for the parking maneuver and speed at which this is executed as well as when a parking spot can be detected in order to run the parking maneuver. We also changed our gestures to be more

clear and recognizable, and changed the manual control parameters in order to use more accurate readings from the LEAP motion controller. These adjustments were to account for the limitations of real-life hardware.

After testing and tuning, the Turtlebot is now able to run through the controlled environment while achieving our system goals. That is, perform autonomous parallel parking while also allowing a human to intervene with its control from a remote work station.

A video of this project with all algorithms functioning is found here: https://drive.google.com/file/d/1lZUKtvfeqe9mcH4OGtyH4-FplmpjByGg/view?usp=sharing

VI. Conclusion

Our algorithm seems to perform really well in the controlled experimental setup. It is worth noting that for the sake of simplicity, a lot of real-life constraints, such as obstacles, were not considered for this project. Incorporating these additional constraints is discussed more in the Future Work section.

A challenge we faced while making the robot move in a straight line during the autonomous phase was that, if the heading angle of the robot was not perpendicular, the robot tended to drift. To fix the issue, we used the ArUco marker on the front wall to continuously correct the robot's heading angle. Another challenge faced was the gesture recognition. We originally were getting incorrect gesture read-outs but fixed this issue by making the gestures more distinct and increasing the size of the training set to make gesture detection more robust.

VII. Future Work

In the future we would like to incorporate more sophisticated vision. This would include detecting the size of the parking space to determine if the space is feasible. We would also like to generate trajectories online based off vision input. Whether this means generating a trajectory that avoids a collision with some object in the domain or to simply adjust the parking maneuver so that it more accurately ends in the center of the spot. A couple other adaptations would be to create a more extensive gesture library with unique actions attributed to each gesture, or a more ambitious adaptation would be to add a manipulator that interacts with objects in the environment. Overall these objectives would add some interesting and useful functionality to the Turtlebot but were unfortunately out of scope for this project.

VIII. Acknowledgements

We would like to thank the University of Minnesota for the use of their facilities and equipment as well as Professor Sattar for lending us the LEAP motion controller. We would also like to thank the many contributors to the ROS packages used for this project such as leap_motion and neuro_gesture_leap. Whom without their extensive framework this project would not have been possible.

References

[1] Powering the world's robots, ROS.org, 2018 Retrieved from: http://www.ros.org/.

[2] OpenCV Library. OpenCV Library, 2018, Retrieved from: http://opencv.org/.

[3] Parhar, Tanvir. Parhartanvir/neuro_gesture_leap. GitHub, July 2015, Retrieved from: http://github.com/parhartanvir/neuro_gesture_leap.git.

[4] Rueckstiess, Thomas. PyBrain. PyBrain, 2010, Retrieved from: http://pybrain.org/.