PackBot

Bharath S, Quinn K, Sowmiya N G
Emails: koeni344@umn.edu, sivar019@umn.edu, govin088@umn.edu

Abstract

Package sorting and delivery within a building can be an arduous and time consuming process. It involves humans sifting through all packages and sometimes making multiple trips if deliveries are staggered. We propose an autonomous solution consisting of a robot identifying, picking up, and dropping off packages. Our implementation consists of a Turtlebot3 matching blocks based on ArUco markers to their respective drop-off locations around a maze. It also incorporates obstacle avoidance to simulate an actual office environment. This system may be scaled up to incorporate reading of package labels and a more advanced grasping mechanism for larger packages. Overall, it mitigates risk of incorrect pick-up and also saves people and businesses time, money, and labor.

Introduction

The boom of e-commerce has steadily increased the amount of packages being delivered, both to corporations and individual customers. Package sorting a delivery within a building of consumers is often a task of its own, requiring people to attend the delivery room, and customers to carefully pick out their packages. However, tools and companies have tried to aid this process using robots for scanning and delivery. One example is the company Light Line Delivery Corp., which has developed robots for contactless delivery in residential high-rises. One such robot is shown in Figure 1.

Figure 1

Keeping the ongoing pandemic in mind, it would be beneficial to have an office robot system capable of collecting and delivering packages to their respective locations in an office building. This would save time and money for the business, while reducing the possible human-human contact that can arise when separating packages.

Though warehouse robotics has gone under much development, the same robots in a warehouse cannot be used in a people-dominated environment. Many issues come up such as identifying package labels, mapping out the building, creating a shortest path to delivery, and more. Our goal is to design a system that can solve these problems at a small test-scale, and then can be modified to be implemented in a real office environment. A combination of SLAM, Vision and simple manipulation will be used.

Problem Description

In this project, the Turtlebot3 will be used to drop-off small labeled blocks from a single pick-up location to their respective drop-off points within a maze. The goal is to create a proof-of-concept that can be scaled and applied in real-life. The general structure of tasks is as follows:

1. Robot traverses the maze and uses SLAM to build a map. We also note the approximate locations of the drop-off points, along with the wall orientation with respect to the robot **[1].** This step builds a map that can be accessed later
2. Robot comes to the pick-up point and scans for packages (blocks)
3. Once blocks are detected, the vision system will pick up the nearest block **[2]**
4. Robot traverses to drop-off point for specified block and avoids collisions on the way **[3]**
5. Robot heads back to pick-up point and repeats until no blocks remain

The requirements of our system are the following:

1. The Robot should create an accurate environment map with interest points (SLAM)
2. The robot should be able to properly identify ArUco markers (1-3) for block and drop-off detection (Image Identification)

Related Work

Robot perception is a widely explored and ever growing field. There are many forms of vision put to use in various applications. For the purpose of our project we are focusing on the detection and localization of a block. Convolutional Neural Networks using deep learning such as YOLO and SSD have shown the effectiveness that single pass CNN's can have with real time object detection. **[9, 10]** . Due to the nature of our project we do need to be able to detect the boxes in real time; however, we have the advantage of

being able to modify the representation of said boxes. In other words, YOLO and SSD are mainly used for multi-class object detection in which the objects come in a range of shapes and sizes and the objects within a class can differ widely. While YOLO and SSD would likely succeed in our application and are even an option to expand on this project to be able to detect many more grabbable objects, we can get away with a much simpler method. This comes in the form of ArUco markers. ArUco is an open source library that allows for the detection of small boxes of black and white squares similar to QR codes **[11]**. These markers come in a variety of dictionaries but all allow for the production of specific markers corresponding to numerical ids, one of the most commonly used dictionaries is described thoroughly in **[13]**. This still leaves the need for, at minimum, a distance measure to the camera. There are a number of methods to accomplish this such as using two cameras, measuring the blurriness of the object, as well as the most common method which uses rotations and translations combined into a homography to represent the transformation from pixel to real space. **[5]** Fortunately for us, with the ArUco library it is possible to estimate the pose of the camera with respect to the marker(s) in focus, if the camera is correctly calibrated. **[11]** This incredible library hits both necessities in one package. An example of this use can be found in **[4]**, in which the authors use ArUco markers for automatic navigation and landing of a drone in indoor environments.

Another important task is building a map of the environment and localizing the robot within it (SLAM). Most systems using LIDAR tend to be a large resource drag on the system due to the vast amount of information being recorded. To combat this, efforts have been made for lightweight navigation such as OrthoSLAM **[6]**. This method only maps perpendicular lines to represent the structure of an environment, resulting in a map with just enough information to navigate. Fortunately, our system will not require sparse mapping because the environments will be fairly simple. However, lightweight SLAM is important to keep in mind for robot products which cannot always have a powerful processor to do classical SLAM computations.

A final consideration is navigating a map with known drop-off points. Ideally, the robot should take the most efficient route from package point to drop-off point. Fortunately, path planning has a long history and is deeply connected with basic search algorithms where the goal is to find a certain item within a set. Namely, A* and B* are the most common algorithms used which find a path to a given goal node that minimizes cost (distance) **[8]**. Though they work well in many cases, the computational complexity for A* is of O(b^d) since it stores all generated nodes in memory. This has allowed newer algorithms to dominate in complex environments, such as rapidly-exploring random trees (RRT) **[7]**. RRT allows for efficient search in high-dimensional space and is

excellent for path-planning and obstacle avoidance. This algorithm would be excellent for dynamic path planning where the objects in our environment are constantly changing and unknown. However, since this project will use controlled environments, we may use the either the A* or B* algorithm. For the sake of simplicity, the Turtlebot SLAM and navigation nodes will be used with some slight alterations.

We note that various efforts have been made by current logistics industry giants towards automating package moving such as Amazon, UPS, and FedEx. One example is the approach to Multi-Agent Path Finding (MAPF) where collision-free paths must be generated for multiple robots given a certain environment **[14].** This creates states of each robot and dependencies via graph networks. This is known as an action dependency graph, where the actions of one node will influence how the other behave. Though this is out of scope for this project, it is a useful method for the case of multiple PackBots where each is designed for a different sized package. This would allow them to interact simultaneously rather than one by one.

Results

ArUco Pose Tracking

Our initial implementation of the ArUco markers was accomplished through the help of the original ArUco documentation **[11]**, as well as associated python documentation **[12]**. ArUco markers come in different dictionaries ranging in number of black and white squares as well as the number of identifiable markers available. For our purposes, a dictionary that is 6x6 with 250 available markers is used. The ArUco library contains functions which allow for easy marker detection and calibration of the camera matrix (containing the x and y focal lengths as well as image width and height) and the distortion matrix by using a ChArUco board-a combination of a chess board and ArUco markers. Only a series of minor changes were made to the original documentation for the implementation to work. The camera matrix and distortion were used in tandem with the corners of the detected markers and associated size of markers to estimate the pose of the camera in relation to each marker. This library function returns two lists of three element vectors, each marker gets a rotation vector and a translation vector representing the transformation from camera to marker. Figures 2 and 3 show the ChArUco board with the identified markers. The green numbers are marker unique id and the blue numbers are distances in centimeters which is found with the L2 norm of the translation vector. The upcoming task is to integrate this with ROS to localize the markers in 3D space with respect to the Turtlebot3.

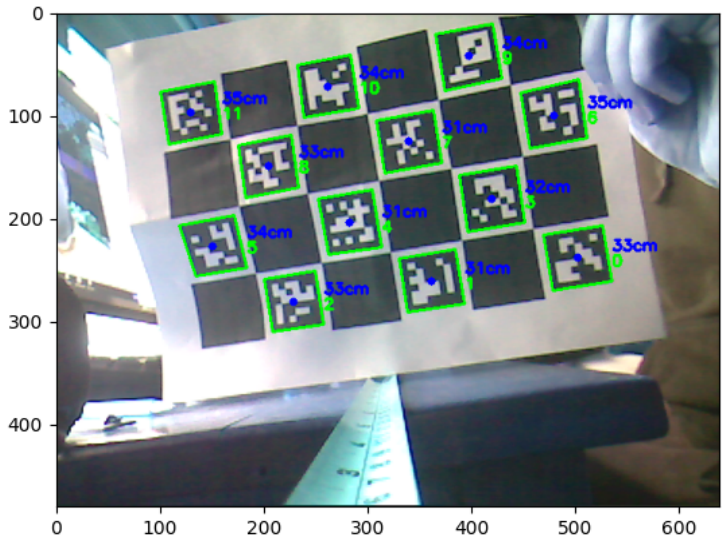

Figure 2

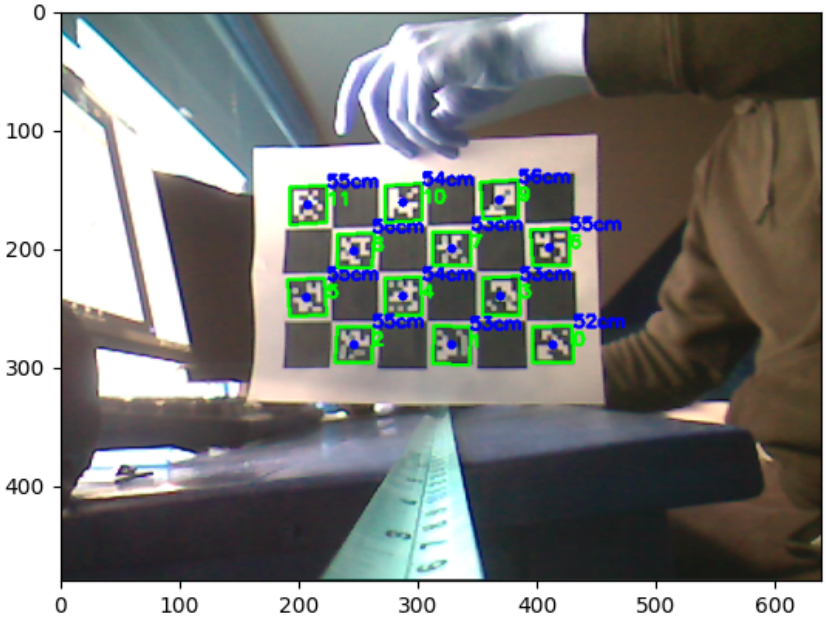

Figure 3

Since this initial implementation we have integrated the ArUco markers with the maze and Turtlebot. The ArUco markers are used for both marking the drop-off and pick up locations as well as identifying the boxes. The following is a description of the process the robot follows to create a map of the environment, locate the ArUco marker drop-off and pick up points, and finally collect and drop-off the ArUco marked packages.

Environment Mapping

Cardboard boxes were used to build a maze which acts as the “office space”. Care was taken to close all gaps between boxes so that the LiDAR doesn’t have unknowns when the bot traverses the maze. For the initial mapping we employed the ROS package Gmapping which utilizes the Turtlebots lidar for simultaneous localization and mapping. This produces a 2D environment map that is used to localize the Turtlebot while moving through the environment. The environment and corresponding map are shown in Figure 4.

Figure 4

Localization of ArUco Marker drop-off and Pick Up Locations

After mapping, the pickup and three drop-off locations were set, as shown in Figure 4. For successful ArUco detection, the point and wall orientation of each location were saved with respect to the global map. This was done by using the teleop function to position the Turtlebot in the desired approximate location and orientation. Each location was saved within a Python dictionary, and orientation was in quaternion form.

Package Pick Up and Delivery

The previous steps are all intended to be run a single time and with teleop to give the robot an accurate internal map of the surroundings and pickup/drop-off locations. Once this has been completed the Turtlebot is then ready to deliver packages and can run autonomously.

This begins with the Turtlebot facing the drop-off location. The node will continuously take in images and detect ArUco marker boxes. If many exist, the bot chooses the

closest box using the L2 norm of the transformation vector. The ID is used to find the corresponding drop location.

The pose of the drop location is passed to the move_base client which allows the Turtlebot to navigate to the desired location. When arriving, it faces in the general direction of the drop-off marker.

The navigation move_base function is utilized to move Turtlebot to the desired locations. Once the Turtlebot has arrived at each location it should be facing the general direction of the drop-off marker. To ensure the best localization, we implement visual servoing using the USB webcam to locate the ArUco marker in frame. The methods described in the earlier ArUco section are used to create a translation matrix from the Turtlebot to the marker with respect to the Turtlebot frame of reference. This is also used to calculate the angle between the two. Twist messages for z-rotation and x-velocity are then published to the bot until goal values are reached. When reached, the package is declared dropped off and the bot backs up and then uses move_base to navigate back to the pick-up location.

Figure 5 shows a Remote PC screenshot of the bot navigating to GoalPoint 1 after scanning the ID. Note that the dictionaries were changed such that ID 3 corresponds to GoalPoint1.

Figure 6 shows the Turtlebot3 burger with the attached cardboard pusher, Logitech USB webcam, and the “package” with the aruco marker.

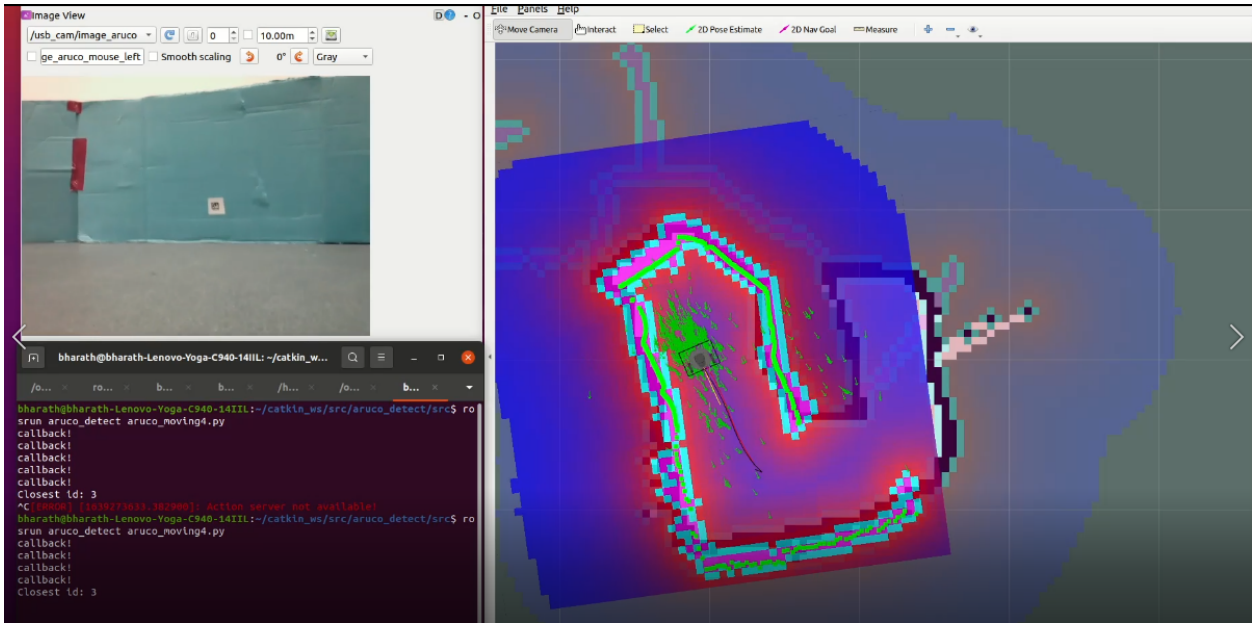

Figure 5

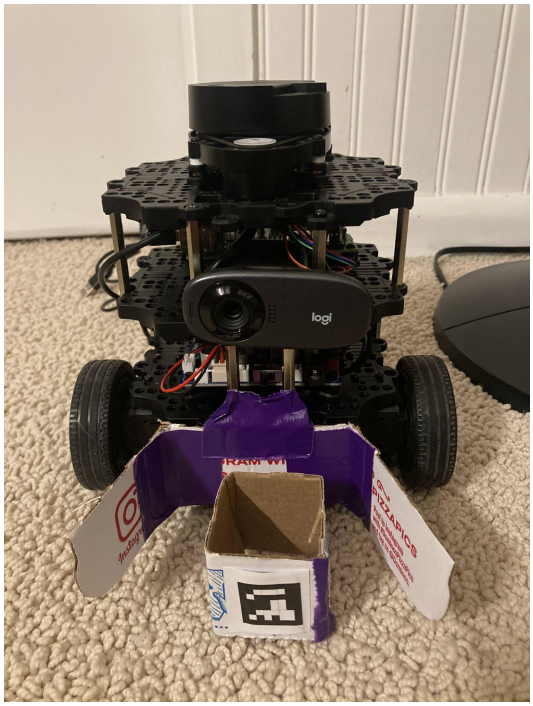

Figure 6

Future Work:

We encountered many difficulties during the implementation of this project. There are many improvements that can be made if work were to continue. This includes automating the two uses of teleop. Teleop is used for initial exploration and map building. This could be replaced with a bug-like algorithm that allows the Turtlebot to explore its environment automatically and build a map simultaneously. The Turtlebot could detect the ArUco markers during this autonomous exploration and use our alignment code to get a good sense of where they are located. This would alleviate the need to manually drive the robot to drop-off locations to get coordinates and an orientation that are then manually hardcoded into a dictionary. This would also allow for the addition of drop-off points with limited work needed for the operators.

An additional area of improvement would be with the move_base function. This function allows us to give the robot a coordinate pair and an orientation and the function handles the path planning to actually get there. Unfortunately this function was also fairly unreliable and often caused the robot to reach its destination and then turn back and forth, overshooting its desired heading and wasting time. Additionally, the padding given to the walls to avoid the robot from hitting them was larger than necessary leading to strange movement executions from the robot as well as the robot deciding to go backwards all the way to its target which is not ideal when it has a package. This could be resolved by implementing our own path finding algorithm such as RRTs using a

PRM. With a custom planning algorithm we could also ensure that the robot always traveled in ways such that a package within its pusher arms would never be dropped by backing up and turning or turning without moving forwards first.

The addition of a mechanical arm for grabbing packages was also discussed but never implemented. This could be made out of 3D printed parts and controlled using servos. Additionally, this arm could be equipped with an electromagnet that could be used to pick up and drop-off packages. This arm would alleviate some movement of the robot which is less accurate than using the real time camera coupled with the arm.

The ultimate goal for this project would be to upscale the robot used so that it would work with larger packages as well as be able to work with packages of various sizes. It would also be interesting to add functionality for multiple robots with swarm mechanics.

Tasks/Timeline

- Week of Nov 7- Finalize Gazebo environments for testing. Make easy to replicate
- Week of Nov 14 - Find out necessary nodes for each task
- Week of Nov 21 - Achieve full image recognition for ArUco detection
- Week of Nov 28 - Finalize code for identifying and going to goal point depending on block marker
- Week of Dec 5 - Make the physical maze and get a map of it. Find the points of interest and save. Then implement node(s)
- Dec 14th (Last Class) - Final Report 10-20 pages, 12pt

Team Management

*The team agrees all work was distributed equally and fairly

Task	Lead
SLAM and LIDAR	Bharath
Node and Remote PC	Bharath
ArUco and Camera	Quinn
Coding and File Manager	Sowmiya
Maze construction	ALL
Research Camera capabilities	Sowmiya
3D Printing	Quinn

Materials

The source code can be found here: https://github.com/GSNCodes/CSCI-5551-PackBot
Please navigate to CSCI-5551-PackBot/aruco_detect/src to view our nodes.

Associated videos and images can be found here: https://drive.google.com/drive/folders/1RP2M6_CbIO8pPdh9ED19S4RdKqb-B2n3?usp=sharing

The presentation can be viewed here: https://docs.google.com/presentation/d/1P8-mfhLdre5FJljBxVGntE0s_zm_Fto0acNznu28N9c/edit?usp=sharing

References

1. Pajaziti, A. (2014). SLAM - Map Building and Navigation via ROS. International Journal of Intelligent Systems and Applications in Engineering, 2(4), 71. doi:10.18201/ijisae.08103 (2014).
2. Born, William K., and Christopher J. Lowrance. "Application of Convolutional Neural Network Image Classification for a Path-Following Robot." *2018 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2018, https://doi.org/10.1109/urtc45901.2018.9244781.
3. L. Tai, S. Li and M. Liu, "A deep-network solution towards model-less obstacle avoidance," 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 2759-2764, doi: 10.1109/IROS.2016.7759428.
4. M. F. Sani and G. Karimian, "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," 2017 International Conference on Computer and Drone Applications (IConDA), 2017, pp. 102-107, doi: 10.1109/ICONDA.2017.8270408.
5. Alizadeh, Peyman. Object distance measurement using a single camera for robotic applications. Diss. Laurentian University of Sudbury, 2015.
    a. https://zone.biblio.laurentian.ca/bitstream/10219/2458/1/Peyman%20Alizadeh%20MSc.%20Thesis%20Corrected_2_2.pdf
6. V. Nguyen, A. Harati, A. Martinelli, R. Siegwart and N. Tomatis, "Orthogonal SLAM: a Step toward Lightweight Indoor Autonomous Navigation," 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 5007-5012, doi: 10.1109/IROS.2006.282527.
7. LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998)
    a. https://www.cs.csustan.edu/~xliang/Courses/CS4710-21S/Papers/06%20RRT.pdf
8. Nosrati, Masoud, Ronak Karimi, and Hojat Allah Hasanvand. "Investigation of the*(star) search algorithms: Characteristics, methods and approaches." World Applied Programming 2.4 (2012): 251-256.
    a. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.685.6608&rep=rep1&type=pdf
9. Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).
    a. https://arxiv.org/pdf/1804.02767.pdf
10. Liu, Wei, et al. "Ssd: Single shot multibox detector." *European conference on computer vision*. Springer, Cham, 2016
    a. https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2
11. ArUco documentation
    a. https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit
12. Python ArUco documentation/examples
    a. https://mecArUco2.readthedocs.io/en/latest/notebooks_rst/ArUco/ArUco.html
13. Garrido-Jurado, Sergio & Muñoz-Salinas, Rafael & Madrid-Cuevas, Francisco & Medina-Carnicer, Rafael. (2015). Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. Pattern Recognition. 51. 10.1016/j.patcog.2015.09.023.
    a. https://www.researchgate.net/publication/282426080_Generation_of_fiducial_marker_dictionaries_using_Mixed_Integer_Linear_Programming
14. Honig, W., Kiesel, S., Tinka, A., Durham, J. W., & Ayanian, N. (2019). Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics and Automation Letters, 4*(2), 1125-1131. doi:10.1109/lra.2019.2894217